Appendice: un sistema giocattolo, denominato «05»

Preparazione 2043
i9.1.1 File-immagine 2044
Directory di lavoro 2045
i9.2.1 Directory «05/» 2045
i9.2.2 Script di collegamento 2050
i9.2.3 Altre directory 2051
Libreria standard per iniziare 2051
i9.3.1 Libreria «limits.h» 2052
File isolati per dichiarazioni riprese in più librerie 2052
i9.4.1 Libreria «stdbool.h» 2054
i9.4.2 Libreria «time.h» 2054
i9.4.3 Libreria «ctype.h» 2055
i9.4.4 Libreria «stdint.h» 2056
i9.4.5 Libreria «inttypes.h» 2059
i9.4.6 Libreria «stdarg.h» 2064
i9.4.7 Libreria «stddef.h» 2065
i9.4.8 Libreria «stdlib.h» 2065
i9.4.9 Libreria «string.h» 2068
i9.4.10 Libreria «stdio.h» 2071
Librerie specifiche generali 2093
i9.5.1 File «build.h» 2093
i9.5.2 Libreria «io.h» 2093
i9.5.3 Libreria «multiboot.h» 2095
i9.5.4 File «os.h» 2097
i9.5.5 Libreria «vga.h» 2101
Un primo kernel di prova 2107
i9.6.1 File «kernel.h» 2107
i9.6.2 Altri file mancanti 2113
i9.6.3 Compilazione e prova di funzionamento 2113

Tabella GDT 2114

i9.7.1 Struttura 2114
i9.7.2 Libreria «gdt.h» 2115
i9.7.3 Modifiche da apportare a «kernel_main.c» 2120

Gestione della memoria 2121

i9.8.1 Gestione della memoria attraverso una lista 2121
i9.8.2 Libreria «mm.h» 2122
i9.8.3 Funzioni per l'allocazione della memoria 2126
i9.8.4 Verifica del funzionamento 2133

Tabella IDT 2135

i9.9.1 File di intestazione «int.h» e file delle routine di interruzione «isr.s» 2135
i9.9.2 Funzioni per definire la tabella IDT 2146
i9.9.3 Gestione delle interruzioni 2152
i9.9.4 Piccole funzioni di contorno 2156
i9.9.5 Verifica del funzionamento 2156

Chiamate di sistema 2157

i9.10.1 File di intestazione «syscall.h» 2158
i9.10.2 Fasi successive all'interruzione 2159
i9.10.3 Verifica del funzionamento 2161

Interruzioni hardware 2162

i9.11.1 Gestione del temporizzatore 2163
i9.11.2 Gestione della tastiera 2165
i9.11.3 Verifica del funzionamento 2170

Una specie di «shell» 2171

i9.12.1 Realizzazione della shell 2172
i9.12.2 Conclusione 2174

`app.h` 2172 `atoi.c` 2065 `bochs` 2045 `build.h` 2093 `cli.s` 2156 `compile` 2045 `ctype.h` 2055 `free.c` 2126 `gdt.c` 2115 `gdt.h` 2115 `gdt_desc_seg.c` 2115 `gdt_load.s` 2115 `gdt_print.c` 2115 `gets.c` 2171 `idt_desc_int.c` 2146 `idt_load.c` 2146 `idt_print.c` 2146 `inb.s`

2093 int.h 2135 inttypes.h 2059 int_128.s 2157 io.h 2093 irq_remap.c 2146 isr.s 2135 isr_exception_name.c 2152 isr_exception_unrecoverable.c 2152 isr_irq.c 2152 2162 isr_syscall.c 2152 2159 kernel.h 2107 kernel_boot.s 2107 kernel_main.c 2174 kernel_memory.c 2107 keyboard.c 2165 keyboard.h 2165 keyboard_load.c 2165 limits.h 2052 linker.ld 2050 makeit 2045 malloc.c 2126 mboot_info.c 2095 mboot_show.c 2095 memcpy.c 2068 memset.c 2068 mm.h 2122 mm_init.c 2122 mm_list.c 2122 mount 2045 multiboot.h 2095 NULL.h 2052 os.h 2097 outb.s 2093 ptrdiff_t.h 2052 realloc.c 2126 restrict.h 2052 shell.c 2172 size_t.h 2052 snprintf.c 2071 stdarg.h 2064 stdbool.h 2054 stddef.h 2065 stdint.h 2056 stdio.h 2071 stdlib.h 2065 sti.s 2156 string.h 2068 strncpy.c 2068 syscall.c 2157 syscall.h 2158 time.h 2054 timer.c 2163 timer.h 2163 timer_freq.c 2163 umount 2045 vga.h 2101 vga_clear.c 2101 vga_new_line.c 2101 vga_printf.c 2101 vga_putc.c 2101 vga_puts.c 2101 vga_set.c 2101 vga_vprintf.c 2101 vsnprintf.c 2071 vsyscall.c 2159 wchar_t.h 2052 _Exit.s 2113

In questa appendice viene descritto il procedimento per realizzare un sistema, estremamente banalizzato, per elaboratori x86-32, sviluppato prima di os16 e di os32. Questo sistema è privo di pianificazione dei processi (*scheduler*), ma soprattutto non è in grado di avviare programmi e nemmeno di accedere a qualche file system. Tuttavia, pur con tutte le sue mancanze, questa specie di sistema può essere utile per comprendere alcuni concetti a chi inizia da zero lo studio delle problematiche connesse con i sistemi operativi.

Giusto per dare un nome a questa cosa, si usa la sigla «05», ovvero le cifre numeriche che più si avvicinano a «os».

Preparazione

Prima di cominciare conviene preparare tutto quello che serve, come viene descritto nelle sezioni successive. Naturalmente ci si avvale degli strumenti di un sistema GNU/Linux per lo sviluppo di questo giocattolo.

Tutti i file descritti in questa appendice dovrebbero essere disponibili a partire da ⟨*allegati/05/*⟩.

File-immagine

«

Per prima cosa serve un file-immagine di un dischetto da 1,44 Mibyte, predisposto con GRUB 1, in modo tale da avviare il file '`kernel`'. In pratica si predispone inizialmente un dischetto reale, con un file system Dos-FAT, si crea la directory '`grub/`' e al suo interno si mettono i file '`stage1`' e '`stage2`' di GRUB 1, assieme al file '`menu.lst`' che può avere semplicemente il contenuto seguente:

```
title kernel
kernel (fd0)/kernel
```

Si mette temporaneamente un file fittizio, denominato '`kernel`', nella directory principale del dischetto e si procede all'installazione del settore di avvio di GRUB 1 stesso:

`#` **`grub`** [*Invio*]

`grub>` **`root (fd0)`** [*Invio*]

```
Filesystem type is fat, using whole disk.
```

`grub>` **`setup (fd0)`** [*Invio*]

```
Checking if "/boot/grub/stage1" exists... no
Checking if "/grub/stage1" exists... yes
Checking if "/grub/stage2" exists... yes
Checking if "/grub/fat_stage1_5" exists... no
Running "install /grub/stage1 (fd0) /grub/stage2 p
  /grub/menu.lst "... succeeded
Done.
```

`grub>` **`quit`** [*Invio*]

A questo punto, avendo terminato il lavoro di installazione di GRUB 1 nel dischetto, si può produrre il file-immagine:

`#` **`cp /dev/fd0 floppy.img`** [*Invio*]

Eventualmente, questo file-immagine, già pronto, dovrebbe essere disponibile da qui: ⟨*allegati/05/floppy.img.gz* ⟩. Tuttavia, prima di usarlo, è necessario decomprimerlo, essendo stato ridotto con GZIP.

Directory di lavoro

Prima di iniziare gli esperimenti, si predispone una directory di lavoro, da utilizzare in qualità di utente comune. Nella directory si copia il file '`floppy.img`' e si mettono alcuni script molto semplici:

Listato u9.4. '`./mount`'

```
#!/bin/sh
chmod a+rw floppy.img
su root -c "mount -o loop,uid=1001 -t vfat floppy.img /mnt/fd0"
```

Listato u9.5. '`./umount`'

```
#!/bin/sh
su root -c "umount /mnt/fd0"
```

Listato u9.6. '`./bochs`'

```
#!/bin/sh
bochs -q 'boot:a' 'floppya: 1_44=floppy.img, status=inserted' 'megs:32'
```

Il senso di questi script è evidente e il loro scopo è solo quello di ridurre al minimo l'impegno di digitazione. In questa directory viene poi predisposto anche lo script '**`compile`**', ma viene descritto nella sezione successiva.

Directory «05/»

A partire dalla directory di lavoro si crea la sottodirectory '`05/`', nella quale viene poi messo il codice del sistema che si va a creare. Ma per evitare di fare confusione con i file-make, si predispone uno script per la compilazione che li crea al volo, in base ai contenuti effettivi delle sottodirectory.

Listato u9.7. '`./05/makeit`'

```
#!/bin/sh
#
# makeit...
#
OPTION="$1"
#
edition () {
    local EDITION="include/kernel/build.h"
    echo -n                               > $EDITION
    echo -n "#define BUILD_DATE \""      >> $EDITION
    echo -n `date "+%Y%m%d%H%M%S"`       >> $EDITION
    echo  "\""  >> $EDITION
}
#
#
```

```
#
makefile () {
    #
    local MAKEFILE="Makefile"
    local TAB="	"
    #
    local SOURCE_C=""
    local C=""
    local SOURCE_S=""
    local S=""
    #
    local c
    local s
    #
    # Trova i file in C.
    #
    for c in *.c
    do
        if [ -f $c ]
        then
        C=`basename $c .c`
            SOURCE_C="$SOURCE_C $C"
        fi
    done
    #
    # Trova i file in ASM.
    #
    for s in *.s
    do
        if [ -f $s ]
        then
        S=`basename $s .s`
            SOURCE_S="$SOURCE_S $S"
        fi
    done
    #
    # Prepara il file make.
    #
    echo -n                                                  > $MAKEFILE
    echo "# Questo file è stato prodotto automaticamente"   >> $MAKEFILE
    echo "# dallo script \"makeit\", sulla base dei"          >> $MAKEFILE
    echo "# contenuti della directory."                       >> $MAKEFILE
    echo "#"                                                  >> $MAKEFILE
    echo "c = $SOURCE_C"                                      >> $MAKEFILE
    echo "#"                                                  >> $MAKEFILE
    echo "s = $SOURCE_S"                                      >> $MAKEFILE
    echo "#"                                                  >> $MAKEFILE
```

```
    echo "all: \$(s) \$(c)"                                     >> $MAKEFILE
    echo "#"                                                    >> $MAKEFILE
    echo "clean:"                                               >> $MAKEFILE
    echo "${TAB}@rm *.o 2> /dev/null ; pwd"                     >> $MAKEFILE
    echo "#"                                                    >> $MAKEFILE
    echo "\$(s):"                                               >> $MAKEFILE
    echo "${TAB}@echo \$@.s"                                    >> $MAKEFILE
    echo "${TAB}@as -o \$@.o \$@.s"                             >> $MAKEFILE
    echo "#"                                                    >> $MAKEFILE
    echo "\$(c):"                                               >> $MAKEFILE
    echo "${TAB}@echo \$@.c"                                    >> $MAKEFILE
    echo "${TAB}@gcc -Wall -Werror -o \$@.o -c \$@.c" \
         "-nostdinc -nostdlib -nostartfiles -nodefaultlibs" \
        "-I../include -I../../include -I../../../include"       >> $MAKEFILE
    #
}
#
#
#
main () {
    #
    local CURDIR=`pwd`
    local OBJECTS
    local d
    local c
    local s
    local o
    #
    edition
    #
    for d in `find .`
    do
        if [ -d "$d" ]
        then
            #
            # Ci sono sorgenti in C o in ASM?
            #
            c=`echo $d/*.c | sed "s/ .*//"`
            s=`echo $d/*.s | sed "s/ .*//"`
            #
            if [ -f "$c" ] || [ -f "$s" ]
            then
                CURDIR=`pwd`
                cd $d
                makefile
                #
                if [ "$OPTION" = "clean" ]
```

```
                then
                    make clean
                else
                    if ! make
                    then
                        cd "$CURDIR"
                        exit
                    fi
                fi
                cd "$CURDIR"
            fi
        fi
    done
    #
    cd "$CURDIR"
    #
    #
    #
    if [ "$OPTION" = "clean" ]
    then
        true
    else
        OBJECTS=""
        #
        for o in `find . -name \*.o -print`
        do
            if    [ "$o" = "./kernel/kernel_boot.o" ] \
               || [ "$o" = "./kernel/kernel_main.o" ] \
               || [ ! -e "$o" ]
            then
                true
            else
                OBJECTS="$OBJECTS $o"
            fi
        done
        #
        echo "Link"
        #
        ld --script=linker.ld -o kernel_image \
                kernel/kernel_boot.o \
                $OBJECTS \
                kernel/kernel_main.o
        #
        cp -f kernel_image /mnt/fd0/kernel
        sync
    fi
}
```

```
#
# Start.
#
if [ -d include ] && [ -d kernel ] && [ -d lib ]
then
    main
else
    echo "Mi trovo in una posizione sbagliata e non posso svolgere" \
         "il mio compito"
fi
```

Va osservato che la variabile '**TAB**' deve contenere esattamente una tabulazione orizzontale (di norma il codice 09_{16}. Pertanto, se si riproduce il file o se lo si scarica, occorre verificare che il contenuto sia effettivamente una tabulazione, altrimenti va corretto. Se la variabile '**TAB**' contiene solo spazi, i file-make che si ottengono non sono validi.

```
    local TAB="	"
```

In pratica, attraverso questo script, i file-make che si generano hanno un aspetto simile a quello del listato seguente:

```
c =	elenco_file_c_senza_estensione
#
s =	elenco_file_asm_senza_estensione
#
all: $(s) $(c)
#
clean:
	@rm *.o 2> /dev/null ; pwd
#
$(s):
	@echo $@.s
	@as -o $@.o $@.s
#
$(c):
	@echo $@.c
	@gcc -Wall -Werror -o $@.o -c $@.c ↩
↪	     -nostdinc -nostdlib -nostartfiles ↩
↪	     -nodefaultlibs -I../include ↩
↪	     -I../../include -I../../../include
```

Il «collegamento» (*link*) dei file avviene attraverso un comando contenuto nello script '**makeit**', dove si fa in modo di mettere all'inizio il file-oggetto che è responsabile dell'avvio, dal momento che contiene l'impronta di riconoscimento per il sistema di avvio aderente alle specifiche *multiboot*.

Nella directory di lavoro descritta nella sezione precedente, conviene mettere uno script che richiami a sua volta '**makeit**' e che provveda a copiare il file del kernel nel file-immagine del dischetto:

Listato u9.9. '`./compile`'

```
#!/bin/sh
cd 05
./makeit clean
./makeit
cd ..
```

Script di collegamento

«

Sempre all'interno della directory '`05/`' va predisposto lo script usato da GNU LD per eseguire correttamente il collegamento dei file oggetto in un file eseguibile unico. Dal momento che nel progetto che si intraprende si intende usare la memoria linearmente, si intende che il blocco minimo sia della dimensione di un registro, ovvero pari a 4 byte:

Listato u9.10. '`./05/linker.ld`'

```
/***********************************************************************
 * La memoria viene usata in modo lineare, senza controlli dei
 * privilegi, così non si usano nemmeno gli allineamenti tradizionali
 * di 4096 byte, ma solo di 4 byte, ovvero di un registro.
 ***********************************************************************/

ENTRY (kernel_boot)
SECTIONS {
    . = 0x00100000;
    k_mem_total_s = .;
    .text : {
        k_mem_text_s = .;
        *(.text)
        . = ALIGN (0x4);
        k_mem_text_e = .;
    }
    .rodata : {
        k_mem_rodata_s = .;
        *(.rodata)
```

```
        . = ALIGN (0x4);
        k_mem_rodata_e = .;
    }
    .data : {
        k_mem_data_s = .;
        *(.data)
        . = ALIGN (0x4);
        k_mem_data_e = .;
    }
    .bss : {
        k_mem_bss_s = .;
        *(.bss)
        *(COMMON)
        . = ALIGN (0x4);
        k_mem_bss_e = .;
    }
    k_mem_total_e = .;
}
```

Il codice contenuto nel file del kernel che si va a produrre, deve iniziare a partire da 00100000_{16}, ovvero da 1 Mibyte, come prescrive il sistema di avvio *multiboot*, il quale va a collocarlo in memoria, a partire da quella posizione. Inoltre, per consentire di individuare i blocchi di memoria utilizzati, vengono inseriti dei simboli; per esempio, '**`k_mem_total_s`**' individua l'inizio del kernel, mentre '**`k_mem_total_e`**' ne individua la fine.

Si dà per scontato che GNU AS predisponga un file eseguibile in formato ELF.

Altre directory

All'interno di '`05/`' si creano ancora: '`lib/`', per la libreria standard e altre librerie specifiche del sistema; '`include/`', per i file di intestazione della libreria; '`kernel/`' con i file iniziali usati dal kernel; '`app/`' per le applicazioni (ovvero le funzioni avviate dal kernel quando tutto è pronto).

Libreria standard per iniziare

Quando si scrive un programma da utilizzare senza l'ausilio del sistema operativo, è necessario realizzare una propria libreria di funzioni C, perché quella che offre il proprio compilatore, è fatta sicuramente per interagire con il sistema operativo che la ospita. Nelle sezioni successive vengono mostrati i file usati nel sistema in corso di presentazione, per una libreria C standard generalizzata.

Va però osservato che possono essere gestiti solo interi con un massimo di 32 bit. Infatti, il compilatore GNU C consentirebbe anche di gestire interi a 64 bit, corrispondenti al tipo '**long long**', ma per questo si avvale di funzioni di libreria non standard che, però, qui non sono state realizzate.

Libreria «limits.h»

«

Il file 'limits.h' dimostra quanto appena accennato a proposito della limitazione nella gestione dei numeri interi. Contrariamente a quanto si fa di solito, i valori sono scritti in esadecimale.

Listato u9.11. './05/include/limits.h'

```
#ifndef _LIMITS_H
#define _LIMITS_H       1

#define CHAR_BIT             (8)
#define SCHAR_MIN        (-0x80)
#define SCHAR_MAX         (0x7F)
#define UCHAR_MAX         (0xFF)
#define CHAR_MIN     SCHAR_MIN
#define CHAR_MAX     SCHAR_MAX
#define MB_LEN_MAX          (16)
#define SHRT_MIN       (-0x8000)
#define SHRT_MAX        (0x7FFF)
#define USHRT_MAX       (0xFFFF)
#define INT_MIN    (-0x80000000)
#define INT_MAX     (0x7FFFFFFF)
#define UINT_MAX    (0xFFFFFFFFU)
#define LONG_MIN   (-0x80000000L)
#define LONG_MAX    (0x7FFFFFFFL)
#define ULONG_MAX   (0xFFFFFFFFUL)

#endif
```

File isolati per dichiarazioni riprese in più librerie

«

Secondo lo standard, più file di libreria dichiarano gli stessi tipi speciali e le stesse costanti. Per evitare confusione, la dichiarazione di queste costanti e di questi tipi condivisi, viene collocata in file isolati che, successivamente, altri file incorporano a seconda della necessità. Inoltre, il compilatore usato per la costruzione di questo si-

stema non gestisce i «puntatori ristretti», ovvero non considera valida la parola chiave '**`restrict`**'. Per mantenere una forma aderente allo standard si aggiunge la dichiarazione della macro-variabile ***restrict*** vuota, in un file separato che molti altri file incorporano.

Listato u9.12. '`./05/include/restrict.h`'

```
#ifndef _RESTRICT_H
#define _RESTRICT_H      1

#define restrict

#endif
```

Listato u9.13. '`./05/include/NULL.h`'

```
#ifndef _NULL_H
#define _NULL_H      1

#define NULL 0

#endif
```

Listato u9.14. '`./05/include/ptrdiff_t.h`'

```
#ifndef _PTRDIFF_T_H
#define _PTRDIFF_T_H    1

typedef long int ptrdiff_t;

#endif
```

Listato u9.15. '`./05/include/size_t.h`'

```
#ifndef _SIZE_T_H
#define _SIZE_T_H       1

typedef unsigned long int size_t;

#endif
```

Listato u9.16. './05/include/wchar_t.h'

```
#ifndef _WCHAR_T_H
#define _WCHAR_T_H      1

typedef unsigned char wchar_t;

#endif
```

Dal file 'wchar_t.h' si comprende che, per il sistema in corso di realizzazione, si intende gestire al massimo la codifica ASCII e nulla di più.

Libreria «stdbool.h»

Listato u9.17. './05/include/stdbool.h'

```
#ifndef _STDBOOL_H
#define _STDBOOL_H      1

#define bool    _Bool
#define true    1
#define false   0
#define __bool_true_false_are_defined   1

#endif
```

Libreria «time.h»

Listato u9.18. './05/include/time.h'

```
#ifndef _TIME_H
#define _TIME_H       1

#include <restrict.h>
#include <size_t.h>
#include <NULL.h>

#define CLOCKS_PER_SEC  100L
typedef long int clock_t;
typedef long int time_t;

struct tm {int tm_sec;  int tm_min;  int tm_hour;
           int tm_mday; int tm_mon;  int tm_year;
```

```
             int tm_wday; int tm_yday; int tm_isdst;};

clock_t     clock      (void);
time_t      time       (time_t *timer);
double      difftime   (time_t time1, time_t time0);
time_t      mktime     (struct tm *timeptr);
struct tm *gmtime      (const time_t *timer);
struct tm *localtime   (const time_t *timer);
char       *asctime    (const struct tm *timeptr);
char       *ctime      (const time_t *timer);
size_t     strftime    (char * restrict s, size_t maxsize,
                        const char * restrict format,
                        const struct tm * restrict timeptr);

#define ctime(t) (asctime (localtime (t)));

#endif
```

Del file 'time.h' viene usato solo il tipo '**clock_t**' e la macro-variabile ***CLOCKS_PER_SEC***, con la quale si dichiara implicitamente la frequenza con cui deve reagire il temporizzatore interno del realizzando sistema. Pertanto, le funzioni del file di cui si vedono i prototipi, non vengono realizzate.

Libreria «ctype.h»

«

Listato u9.19. './05/include/ctype.h'

```
#ifndef _CTYPE_H
#define _CTYPE_H        1

#include <NULL.h>

#define isblank(C)   ((int) (C == ' ' || C == '\t'))
#define isspace(C)   ((int) (C == ' ' \
                            || C == '\f' \
                            || C == '\n' \
                            || C == '\n' \
                            || C == '\r' \
                            || C == '\t' \
                            || C == '\v'))
```

```
#define isdigit(C)   ((int) (C >= '0' && C <= '9'))
#define isxdigit(C)  ((int) ((C >= '0' && C <= '9') \
                        || (C >= 'A' && C <= 'F') \
                        || (C >= 'a' && C <= 'f')))
#define isupper(C)   ((int) (C >= 'A' && C <= 'Z'))
#define islower(C)   ((int) (C >= 'a' && C <= 'z'))
#define iscntrl(C)   \
    ((int) ((C >= 0x00 && C <= 0x1F) || C == 0x7F))
#define isgraph(C)   ((int) (C >= 0x21 && C <= 0x7E))
#define isprint(C)   ((int) (C >= 0x20 && C <= 0x7E))
#define isalpha(C)   (isupper (C) || islower (C))
#define isalnum(C)   (isalpha (C) || isdigit (C))
#define ispunct(C)   \
    (isgraph (C) && (!isspace (C)) && (!isalnum (C)))

#endif
```

Libreria «stdint.h»

«

Listato u9.20. './05/include/stdint.h'

```
#ifndef _STDINT_H
#define _STDINT_H       1

typedef signed char                 int8_t;
typedef short int                  int16_t;
typedef int                        int32_t;      // x86-32
typedef unsigned char              uint8_t;
typedef unsigned short int        uint16_t;
typedef unsigned int              uint32_t;      // x86-32

#define INT8_MIN                    (-0x80)
#define INT8_MAX                     (0x7F)
#define UINT8_MAX                    (0xFF)
#define INT16_MIN                 (-0x8000)
#define INT16_MAX                  (0x7FFF)
#define UINT16_MAX                 (0xFFFF)
#define INT32_MIN             (-0x80000000)
#define INT32_MAX              (0x7FFFFFFF)
```

```
#define UINT32_MAX                  (0xFFFFFFFFU)

typedef signed char                int_least8_t;
typedef short int                  int_least16_t;
typedef int                        int_least32_t;
typedef unsigned char              uint_least8_t;
typedef unsigned short int         uint_least16_t;
typedef unsigned int               uint_least32_t;

#define INT_LEAST8_MIN                   (-0x80)
#define INT_LEAST8_MAX                    (0x7F)
#define UINT_LEAST8_MAX                   (0xFF)
#define INT_LEAST16_MIN                (-0x8000)
#define INT_LEAST16_MAX                 (0x7FFF)
#define UINT_LEAST16_MAX                (0xFFFF)
#define INT_LEAST32_MIN            (-0x80000000)
#define INT_LEAST32_MAX             (0x7FFFFFFF)
#define UINT_LEAST32_MAX           (0xFFFFFFFFU)

#define INT8_C(VAL)      VAL
#define INT16_C(VAL)     VAL
#define INT32_C(VAL)     VAL
#define UINT8_C(VAL)     VAL
#define UINT16_C(VAL)    VAL
#define UINT32_C(VAL)    VAL ## U

typedef signed char                 int_fast8_t;
typedef int                         int_fast16_t;
typedef int                         int_fast32_t;
typedef unsigned char               uint_fast8_t;
typedef unsigned int               uint_fast16_t;
typedef unsigned int               uint_fast32_t;

#define INT_FAST8_MIN                    (-0x80)
#define INT_FAST8_MAX                     (0x7F)
#define UINT_FAST8_MAX                    (0xFF)
#define INT_FAST16_MIN             (-0x80000000)
#define INT_FAST16_MAX              (0x7FFFFFFF)
```

```
#define UINT_FAST16_MAX          (0xFFFFFFFFU)
#define INT_FAST32_MIN          (-0x80000000)
#define INT_FAST32_MAX           (0x7FFFFFFF)
#define UINT_FAST32_MAX          (0xFFFFFFFFU)

typedef int                          intptr_t;
typedef unsigned int                uintptr_t;

#define INTPTR_MIN              (-0x80000000)
#define INTPTR_MAX              (0x7FFFFFFF)
#define UINTPTR_MAX             (0xFFFFFFFFU)

typedef long int                    intmax_t;
typedef unsigned long int          uintmax_t;

#define INTMAX_C(VAL)   VAL ## L
#define UINTMAX_C(VAL)  VAL ## UL

#define INTMAX_MIN             (-0x80000000L)
#define INTMAX_MAX              (0x7FFFFFFFL)
#define UINTMAX_MAX             (0xFFFFFFFFUL)

#define PTRDIFF_MIN            (-0x80000000)
#define PTRDIFF_MAX             (0x7FFFFFFF)

#define SIG_ATOMIC_MIN         (-0x80000000)
#define SIG_ATOMIC_MAX          (0x7FFFFFFF)

#define SIZE_MAX                (0xFFFFFFFFU)

#define WCHAR_MIN                        (0)
#define WCHAR_MAX                    (0xFFFFU)

#define WINT_MIN                    (-0x8000L)
#define WINT_MAX                     (0x7FFFL)

#endif
```

Libreria «inttypes.h»

Listato u9.21. './05/include/inttypes.h'

```
#ifndef _INTTYPES_H
#define _INTTYPES_H     1

#include <restrict.h>
#include <stdint.h>
#include <wchar_t.h>

typedef struct {intmax_t quot; intmax_t rem;} imaxdiv_t;

#define PRId8           "d"
#define PRId16          "d"
#define PRId32          "d"
#define PRId64          "lld"
#define PRIdLEAST8      "d"
#define PRIdLEAST16     "d"
#define PRIdLEAST32     "d"
#define PRIdLEAST64     "lld"
#define PRIdFAST8       "d"
#define PRIdFAST16      "d"
#define PRIdFAST32      "d"
#define PRIdFAST64      "lld"
#define PRIdMAX         "lld"
#define PRIdPTR         "d"
#define PRIi8           "i"
#define PRIi16          "i"
#define PRIi32          "i"
#define PRIi64          "lli"
#define PRIiLEAST8      "i"
#define PRIiLEAST16     "i"
#define PRIiLEAST32     "i"
#define PRIiLEAST64     "lli"
#define PRIiFAST8       "i"
#define PRIiFAST16      "i"
#define PRIiFAST32      "i"
#define PRIiFAST64      "lli"
#define PRIiMAX         "lli"
```

```
#define PRIiPTR          "i"
#define PRIb8            "b"      // PRIb... non è standard
#define PRIb16           "b"      //
#define PRIb32           "b"      //
#define PRIb64           "llb"    //
#define PRIbLEAST8       "b"      //
#define PRIbLEAST16      "b"      //
#define PRIbLEAST32      "b"      //
#define PRIbLEAST64      "llb"    //
#define PRIbFAST8        "b"      //
#define PRIbFAST16       "b"      //
#define PRIbFAST32       "b"      //
#define PRIbFAST64       "llb"    //
#define PRIbMAX          "llb"    //
#define PRIbPTR          "b"      //
#define PRIo8            "o"
#define PRIo16           "o"
#define PRIo32           "o"
#define PRIo64           "llo"
#define PRIoLEAST8       "o"
#define PRIoLEAST16      "o"
#define PRIoLEAST32      "o"
#define PRIoLEAST64      "llo"
#define PRIoFAST8        "o"
#define PRIoFAST16       "o"
#define PRIoFAST32       "o"
#define PRIoFAST64       "llo"
#define PRIoMAX          "llo"
#define PRIoPTR          "o"
#define PRIu8            "u"
#define PRIu16           "u"
#define PRIu32           "u"
#define PRIu64           "llu"
#define PRIuLEAST8       "u"
#define PRIuLEAST16      "u"
#define PRIuLEAST32      "u"
#define PRIuLEAST64      "llu"
#define PRIuFAST8        "u"
```

```
#define PRIuFAST16       "u"
#define PRIuFAST32       "u"
#define PRIuFAST64       "llu"
#define PRIuMAX          "llu"
#define PRIuPTR          "u"
#define PRIx8            "x"
#define PRIx16           "x"
#define PRIx32           "x"
#define PRIx64           "llx"
#define PRIxLEAST8       "x"
#define PRIxLEAST16      "x"
#define PRIxLEAST32      "x"
#define PRIxLEAST64      "llx"
#define PRIxFAST8        "x"
#define PRIxFAST16       "x"
#define PRIxFAST32       "x"
#define PRIxFAST64       "llx"
#define PRIxMAX          "llx"
#define PRIxPTR          "x"
#define PRIX8            "X"
#define PRIX16           "X"
#define PRIX32           "X"
#define PRIX64           "llX"
#define PRIXLEAST8       "X"
#define PRIXLEAST16      "X"
#define PRIXLEAST32      "X"
#define PRIXLEAST64      "llX"
#define PRIXFAST8        "X"
#define PRIXFAST16       "X"
#define PRIXFAST32       "X"
#define PRIXFAST64       "llX"
#define PRIXMAX          "llX"
#define PRIXPTR          "X"

#define SCNd8            "hhd"
#define SCNd16           "hd"
#define SCNd32           "d"
#define SCNd64           "lld"
```

```
#define SCNdLEAST8      "hhd"
#define SCNdLEAST16     "hd"
#define SCNdLEAST32     "d"
#define SCNdLEAST64     "lld"
#define SCNdFAST8       "hhd"
#define SCNdFAST16      "d"
#define SCNdFAST32      "d"
#define SCNdFAST64      "lld"
#define SCNdMAX         "lld"
#define SCNdPTR         "d"
#define SCNi8           "hhi"
#define SCNi16          "hi"
#define SCNi32          "i"
#define SCNi64          "lli"
#define SCNiLEAST8      "hhi"
#define SCNiLEAST16     "hi"
#define SCNiLEAST32     "i"
#define SCNiLEAST64     "lli"
#define SCNiFAST8       "hhi"
#define SCNiFAST16      "i"
#define SCNiFAST32      "i"
#define SCNiFAST64      "lli"
#define SCNiMAX         "lli"
#define SCNiPTR         "i"
#define SCNb8           "hhb"   // SCNb... non è standard
#define SCNb16          "hb"    //
#define SCNb32          "b"     //
#define SCNb64          "llb"   //
#define SCNbLEAST8      "hhb"   //
#define SCNbLEAST16     "hb"    //
#define SCNbLEAST32     "b"     //
#define SCNbLEAST64     "llb"   //
#define SCNbFAST8       "hhb"   //
#define SCNbFAST16      "b"     //
#define SCNbFAST32      "b"     //
#define SCNbFAST64      "llb"   //
#define SCNbMAX         "llb"   //
#define SCNbPTR         "b"     //
```

```
#define SCNo8             "hho"
#define SCNo16            "ho"
#define SCNo32            "o"
#define SCNo64            "llo"
#define SCNoLEAST8        "hho"
#define SCNoLEAST16       "ho"
#define SCNoLEAST32       "o"
#define SCNoLEAST64       "llo"
#define SCNoFAST8         "hho"
#define SCNoFAST16        "o"
#define SCNoFAST32        "o"
#define SCNoFAST64        "llo"
#define SCNoMAX           "llo"
#define SCNoPTR           "o"
#define SCNu8             "hhu"
#define SCNu16            "hu"
#define SCNu32            "u"
#define SCNu64            "llu"
#define SCNuLEAST8        "hhu"
#define SCNuLEAST16       "hu"
#define SCNuLEAST32       "u"
#define SCNuLEAST64       "llu"
#define SCNuFAST8         "hhu"
#define SCNuFAST16        "u"
#define SCNuFAST32        "u"
#define SCNuFAST64        "llu"
#define SCNuMAX           "llu"
#define SCNuPTR           "u"
#define SCNx8             "hhx"
#define SCNx16            "hx"
#define SCNx32            "x"
#define SCNx64            "llx"
#define SCNxLEAST8        "hhx"
#define SCNxLEAST16       "hx"
#define SCNxLEAST32       "x"
#define SCNxLEAST64       "llx"
#define SCNxFAST8         "hhx"
#define SCNxFAST16        "x"
```

```
#define SCNxFAST32      "x"
#define SCNxFAST64      "llx"
#define SCNxMAX         "llx"
#define SCNxPTR         "x"

imaxdiv_t imaxdiv    (intmax_t numer, intmax_t denom);
intmax_t  strtoimax  (const char *restrict nptr,
                      char **restrict endptr,
                      int base);
uintmax_t strtouimax (const char *restrict nptr,
                      char **restrict endptr,
                      int base);
intmax_t  wcstoimax  (const wchar_t *restrict nptr,
                      wchar_t **restrict endptr,
                      int base);
uintmax_t wcstouimax (const wchar_t *restrict nptr,
                      wchar_t **restrict endptr,
                      int base);
#endif
```

La libreria '`inttypes.h`' serve per le macro-variabili del tipo '**`PRI`*xn***', in modo da utilizzare correttamente la funzione ***printf()***, mentre si fa riferimento a tipi di valori numerici definiti nel file '`stdint.h`'. Pertanto, le funzioni non vengono realizzate.

Libreria «stdarg.h»

«

Listato u9.22. './05/include/stdarg.h'

```
#ifndef _STDARG_H
#define _STDARG_H       1

typedef unsigned char *va_list;

#define va_start(ap, last) ((void) ((ap) = \
    ((va_list) &(last)) + (sizeof (last))))
#define va_end(ap)         ((void) ((ap) = 0))
#define va_copy(dest, src) \
    ((void) ((dest) = (va_list) (src)))
#define va_arg(ap, type)   \
    (((ap) = (ap) + (sizeof (type))), \
        *((type *) ((ap) - (sizeof (type)))))
#endif
```

Libreria «stddef.h»

Listato u9.23. './05/include/stddef.h'

```
#ifndef _STDDEF_H
#define _STDDEF_H      1

#include <ptrdiff_t.h>
#include <size_t.h>
#include <wchar_t.h>
#include <NULL.h>

#define offsetof(type, member)   \
    ((size_t) &((type *)0)->member)

#endif
```

Libreria «stdlib.h»

Listato u9.24. './05/include/stdlib.h'

```
#ifndef _STDLIB_H
#define _STDLIB_H       1

#include <size_t.h>
#include <wchar_t.h>
```

```
#include <NULL.h>
#include <limits.h>
#include <restrict.h>

typedef struct {int quot; int rem;} div_t;
typedef struct {long int quot; long int rem;} ldiv_t;
typedef struct {long long int quot; long long int rem;} lldiv_t;

#define EXIT_FAILURE    1
#define EXIT_SUCCESS    0
#define RAND_MAX        INT_MAX
#define MB_CUR_MAX      ((size_t) MB_LEN_MAX)

int             atoi  (const char *nptr);
long int        atol  (const char *nptr);
long long int   atoll (const char *nptr);
double          atof  (const char *nptr);

float                  strtof   (const char * restrict nptr,
                                 char ** restrict endptr);
double                 strtod   (const char * restrict nptr,
                                 char ** restrict endptr);
long double            strtold  (const char * restrict nptr,
                                 char ** restrict endptr);
long int               strtol   (const char * restrict nptr,
                                 char ** restrict endptr, int base);
long long int          strtoll  (const char * restrict nptr,
                                 char ** restrict endptr, int base);
unsigned long int      strtoul  (const char * restrict nptr,
                                 char ** restrict endptr, int base);
unsigned long long int strtoull (const char * restrict nptr,
                                 char ** restrict endptr, int base);

int   rand    (void);
void  srand   (unsigned int seed);

void *malloc  (size_t size);
void *realloc (void *ptr, size_t size);
void  free    (void *ptr);
#define calloc(nmemb, size) (malloc ((nmemb) * (size)))

int   atexit  (void (*func) (void));
void  exit    (int status);
void  _Exit   (int status);
void  abort   (void);
```

```
char *getenv  (const char *name);
int   system  (const char *string);

void  qsort   (void *base,
               size_t nmemb,
               size_t size,
               int (*compar) (const void *, const void *));
void *bsearch (const void *key,
               const void *base,
               size_t nmemb,
               size_t size,
               int (*compar) (const void *, const void *));

int abs              (int j);
long int labs        (long int j);
long long int llabs (long long int j);

div_t   div   (int numer, int denom);
ldiv_t  ldiv  (long int numer, long int denom);
lldiv_t lldiv (long long int numer, long long int denom);

int    mblen    (const char *s, size_t n);
int    mbtowc   (wchar_t *restrict pwc, const char *restrict s, size_t n);
int    wctomb   (char *s, wchar_t wc);
size_t mbstowcs (wchar_t *restrict pwcs, const char *restrict s, size_t n);
size_t wcstombs (char *restrict s, const wchar_t *restrict pwcs, size_t n);

#endif
```

Di questa libreria vengono realizzate solo alcune funzioni, ma in particolare, ***_Exit()***, ***malloc()***, ***realloc()*** e ***free()***, dipendono strettamente dal contesto del sistema; pertanto vengono mostrate a parte, in un'altra sezione più specifica.

Listato u9.25. '`./05/lib/atoi.c`'

```
#include <stdlib.h>
#include <ctype.h>
int
atoi (const char *nptr)
{
    int i;
    int sign = +1;
    int n;

    for (i = 0; isspace (nptr[i]); i++)
```

```
    {
      ;       // Si limita a saltare gli spazi iniziali.
    }

  if (nptr[i] == '+')
    {
      sign = +1;
      i++;
    }
  else if (nptr[i] == '-')
    {
      sign = -1;
      i++;
    }

  for (n = 0; isdigit (nptr[i]); i++)
    {
      // Accumula il valore.
      n = (n * 10) + (nptr[i] - '0');
    }

  return sign * n;
}
```

Libreria «string.h»

«

Listato u9.26. './05/include/string.h'

```
#ifndef _STRING_H
#define _STRING_H       1

#include <restrict.h>
#include <size_t.h>
#include <NULL.h>

void *memcpy  (void *restrict dst, const void *restrict org,
               size_t n);
void *memmove (void *dst, const void *org, size_t n);
```

```
char *strcpy  (char *restrict dst,
               const char *restrict org);
char *strncpy (char *restrict dst, const char *restrict org,
               size_t n);
char *strcat  (char *restrict dst, const char *restrict org);
char *strncat (char *restrict dst, const char *restrict org,
               size_t n);

int    memcmp  (const void *s1, const void *s2, size_t n);
int    strcmp  (const char *s1, const char *s2);
int    strcoll (const char *s1, const char *s2);
int    strncmp (const char *s1, const char *s2, size_t n);
size_t strxfrm (char *restrict dst,
                const char *restrict org, size_t n);

void  *memchr  (const void *s, int c, size_t n);
char  *strchr  (const char *s, int c);
char  *strrchr (const char *s, int c);
size_t strspn  (const char *s, const char *accept);
size_t strcspn (const char *s, const char *reject);
char  *strpbrk (const char *s, const char *accept);
char  *strstr  (const char *string, const char *substring);
char  *strtok  (char *restrict string,
                const char *restrict delim);

void  *memset   (void *s, int c, size_t n);
char  *strerror (int errnum);
size_t strlen   (const char *s);

#endif
```

Delle funzioni dichiarate nel file ‘`string.h`’ vengono realizzate solo quelle dei listati successivi.

Listato u9.27. '`./05/lib/memset.c`'

```
#include <string.h>
void
*memset (void *s, int c, size_t n)
{
    unsigned char *a = (unsigned char *) s;
    unsigned char x  = (unsigned char) c;
    size_t i;
    for (i = 0; n > 0 && i < n; i++)
      {
        a[i] = x;
      }
    return s;
}
```

Listato u9.28. '`./05/lib/strncpy.c`'

```
#include <string.h>
char
*strncpy (char *restrict dst, const char *restrict org,
          size_t n)
{
    size_t i;
    for (i = 0; n > 0 && i < n && org[i] != 0; i++)
      {
        dst[i] = org[i];
      }
    for ( ; n > 0 && i < n; i++)
      {
        dst[i] = 0;
      }
    return dst;
}
```

Listato u9.29. './05/lib/memcpy.c'

```
#include <string.h>
void *
memcpy (void *restrict dst, const void *restrict org,
        size_t n)
{
    unsigned char *d = (unsigned char *) dst;
    unsigned char *o = (unsigned char *) org;
    size_t i;
    for (i = 0; n > 0 && i < n; i++)
      {
        d[i] = o[i];
      }
    return dst;
}
```

Libreria «stdio.h»

La libreria che è rappresentata dal file 'stdio.h' è la più noiosa di questo gruppo iniziale. Qui viene mostrato un file incompleto, contenente solo ciò che serve al sistema in corso di realizzazione.

Listato u9.30. './05/include/stdio.h'

```
#ifndef _STDIO_H
#define _STDIO_H        1

#include <restrict.h>
#include <size_t.h>
#include <stdarg.h>
#include <stdint.h>
#include <kernel/vga.h>

int vsnprintf (char *restrict s, size_t n,
               const char *restrict format, va_list arg);
int snprintf  (char *restrict s, size_t n,
               const char *restrict format, ...);

#define vsnprintf(s, n, format, arg) (vsnprintf (s, n, format, arg))
#define vsprintf(s, format, arg)     (vsnprintf (s, SIZE_MAX, format, arg))
#define sprintf(s, ...)              (snprintf (s, SIZE_MAX, __VA_ARGS__))

#define vprintf(format, arg)         (vga_vprintf (format, arg))
```

```
#define printf(...)                    (vga_printf (__VA_ARGS__))
#define puts(s)                        (vga_puts (s, SIZE_MAX) ; \
                                        vga_puts ("\n", 2))
#define putchar(c)                     (vga_putc (c), c)

char *gets (char *s);

// Il resto del file «stdio.h» standard viene omesso.
#endif
```

Le uniche funzioni che si possono realizzare in modo generalizzato sono ***vsnprintf()*** e ***snprintf()***; tuttavia, la realizzazione che viene mostrata è incompleta, in quanto si consente solo la visualizzazione di numeri interi e stringhe. Nel listato successivo, relativo a '`vsnprintf.c`', si vedono diverse funzioni dichiarate in modo «statico», dato che servono esclusivamente a ***vsnprintf()***.

Listato u9.31. '`./05/lib/vsnprintf.c`'

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
//
// Converte un intero senza segno di rango massimo in una stringa.
//
static size_t
uimaxtoa (uintmax_t integer, char *buffer, int base, int uppercase, size_t n)
{
        uintmax_t  integer_copy = integer;
           size_t  digits;
              int  b;
    unsigned char  remainder;

    for (digits = 0; integer_copy > 0; digits++)
      {
        integer_copy = integer_copy / base;
      }

    if (buffer == NULL && integer == 0) return 1;
    if (buffer == NULL && integer > 0)  return digits;

    if (integer == 0)
      {
        buffer[0] = '0';
        buffer[1] = '\0';
        return 1;
      }
```

```
    if (n > 0 && digits > n) digits = n;  // Sistema il numero massimo
                                          // di cifre.

    *(buffer + digits) = '\0';            // Fine della stringa.

    for (b = digits - 1; integer != 0 && b >= 0; b--)
      {
        remainder = integer % base;
        integer   = integer / base;

        if (remainder <= 9)
          {
            *(buffer + b) = remainder + '0';
          }
        else
          {
            if (uppercase)
              {
                *(buffer + b) = remainder - 10 + 'A';
              }
            else
              {
                *(buffer + b) = remainder - 10 + 'a';
              }
          }
      }
    return digits;
}
//
// Converte un intero con segno di rango massimo in una stringa.
//
static size_t
imaxtoa (intmax_t integer, char *buffer, int base, int uppercase, size_t n)
{

    if (integer >= 0)
      {
        return uimaxtoa (integer, buffer, base, uppercase, n);
      }
    //
    // A questo punto c'è un valore negativo, inferiore a zero.
    //
    if (buffer == NULL)
      {
        return uimaxtoa (-integer, NULL, base, uppercase, n) + 1;
      }

    *buffer = '-';                 // Serve il segno meno all'inizio.
```

```
    if (n == 1)
      {
        *(buffer + 1) = '\0';
        return 1;
      }
    else
      {
        return uimaxtoa (-integer, buffer+1, base, uppercase, n-1) + 1;
      }
}
//
// Converte un intero con segno di rango massimo in una stringa,
// mettendo il segno anche se è positivo.
//
static size_t
simaxtoa (intmax_t integer, char *buffer, int base, int uppercase, size_t n)
{

    if (buffer == NULL && integer >= 0)
      {
        return uimaxtoa (integer, NULL, base, uppercase, n) + 1;
      }

    if (buffer == NULL && integer < 0)
      {
        return uimaxtoa (-integer, NULL, base, uppercase, n) + 1;
      }
    //
    // A questo punto «buffer» è diverso da NULL.
    //
    if (integer >= 0)
      {
        *buffer = '+';
      }
    else
      {
        *buffer = '-';
      }

    if (n == 1)
      {
        *(buffer + 1) = '\0';
        return 1;
      }

    if (integer >= 0)
      {
        return uimaxtoa (integer, buffer+1, base, uppercase, n-1) + 1;
      }
```

```
    else
      {
        return uimaxtoa (-integer, buffer+1, base, uppercase, n-1) + 1;
      }
}
//
// Converte un intero senza segno di rango massimo in una stringa,
// provvedendo a sistemare anche l'allineamento.
//
static size_t
uimaxtoa_fill (uintmax_t integer, char *buffer, int base,
               int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = uimaxtoa (integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return uimaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
      {
        return uimaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return uimaxtoa (integer, buffer, base, uppercase, abs (width));
      }
    //
    // size_i <= abs (width).
    //
    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Allineamento a sinistra.
        uimaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Allineamento a destra.
```

```
        memset (buffer, filler, size_f);
        uimaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Converte un intero con segno di rango massimo in una stringa,
// provvedendo a sistemare anche l'allineamento.
//
static size_t
imaxtoa_fill (intmax_t integer, char *buffer, int base,
              int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = imaxtoa (integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return imaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
      {
        return imaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return imaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    // size_i <= abs (width).

    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Allineamento a sinistra.
        imaxtoa (integer, buffer, base, uppercase, 0);
        memset (buffer + size_i, filler, size_f);
      }
    else
```

```
      {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        imaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Converte un intero con segno di rango massimo in una stringa,
// mettendo il segno anche se è positivo, provvedendo a sistemare
// l'allineamento.
//
static size_t
simaxtoa_fill (intmax_t integer, char *buffer, int base,
               int uppercase, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_i = simaxtoa (integer, NULL, base, uppercase, 0);
    size_t size_f;

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (size_i > abs (width))
      {
        return simaxtoa (integer, buffer, base, uppercase, abs (width));
      }

    if (width == 0 && max > 0)
      {
        return simaxtoa (integer, buffer, base, uppercase, max);
      }

    if (width == 0)
      {
        return simaxtoa (integer, buffer, base, uppercase, abs (width));
      }
    //
    // size_i <= abs (width).
    //
    size_f = abs (width) - size_i;

    if (width < 0)
      {
        // Allineamento a sinistra.
        simaxtoa (integer, buffer, base, uppercase, 0);
```

```
        memset (buffer + size_i, filler, size_f);
      }
    else
      {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        simaxtoa (integer, buffer + size_f, base, uppercase, 0);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// Trasferisce una stringa provvedendo all'allineamento.
//
static size_t
strtostr_fill (char *string, char *buffer, int width, int filler, int max)
{
    if (max < 0) return 0; // «max» deve essere un valore positivo.

    size_t size_s = strlen (string);
    size_t size_f;

    if (width > 0 &&  max > 0 && width >  max) width =  max;
    if (width < 0 && -max < 0 && width < -max) width = -max;

    if (width != 0 && size_s > abs (width))
      {
        memcpy (buffer, string, abs (width));
        buffer[width] = '\0';
        return width;
      }

    if (width == 0 && max > 0 && size_s > max)
      {
        memcpy (buffer, string, max);
        buffer[max] = '\0';
        return max;
      }

    if (width == 0 && max > 0 && size_s < max)
      {
        memcpy (buffer, string, size_s);
        buffer[size_s] = '\0';
        return size_s;
      }
    //
    // width =! 0
    // size_s <= abs (width)
```

```
    //
    size_f = abs (width) - size_s;

    if (width < 0)
      {
        // Allineamento a destra.
        memset (buffer, filler, size_f);
        strncpy (buffer+size_f, string, size_s);
      }
    else
      {
        // Allineamento a sinistra.
        strncpy (buffer, string, size_s);
        memset (buffer+size_s, filler, size_f);
      }
    *(buffer + abs (width)) = '\0';

    return abs (width);
}
//
// La funzione «vsnprintf()»
//
int
vsnprintf (char *restrict string, size_t n,
           const char *restrict format, va_list ap)
{

    if (n > INT_MAX) n = INT_MAX;        // «n» non può essere superiore
                                         // a INT_MAX.
    //
    // Al massimo si producono "n-1" caratteri, + '\0'.
    // "n" viene usato anche come dimensione massima per le
    // stringhe interne, se non è troppo grande.
    //
    int         f                    = 0;
    int         s                    = 0;
    int         remain               = n - 1;

    bool        specifier            = 0;
    bool        specifier_flags      = 0;
    bool        specifier_width      = 0;
    bool        specifier_precision  = 0;
    bool        specifier_type       = 0;

    bool        flag_plus            = 0;
    bool        flag_minus           = 0;
    bool        flag_space           = 0;
    bool        flag_alternate       = 0;
    bool        flag_zero            = 0;
```

```
    int         alignment;
    int         filler;

    intmax_t    value_i;
    uintmax_t   value_ui;
    char       *value_cp;

    size_t      width;
    size_t      precision;
    size_t      str_size = n > 1024 ? 1024 : n;
    char        width_string[str_size];
    char        precision_string[str_size];
    int         w;
    int         p;

    width_string[0]     = '\0';
    precision_string[0] = '\0';

    while (format[f] != 0 && s < (n - 1))
      {
        if (!specifier)
          {
            if (format[f] != '%')
              {
                string[s] = format[f];
                s++;
                remain--;
                f++;
                continue;
              }
            if (format[f] == '%' && format[f+1] == '%')
              {
                string[s] = '%';
                f++;
                f++;
                s++;
                remain--;
                continue;
              }
            if (format[f] == '%')
              {
                f++;
                specifier = 1;
                specifier_flags = 1;
                continue;
              }
          }
```

```
        if (specifier && specifier_flags)
          {
            if (format[f] == '+')
              {
                flag_plus = 1;
                f++;
                continue;
              }
            else if (format[f] == '-')
              {
                flag_minus = 1;
                f++;
                continue;
              }
            else if (format[f] == ' ')
              {
                flag_space = 1;
                f++;
                continue;
              }
            else if (format[f] == '#')
              {
                flag_alternate = 1;
                f++;
                continue;
              }
            else if (format[f] == '0')
              {
                flag_zero = 1;
                f++;
                continue;
              }
            else
              {
                specifier_flags = 0;
                specifier_width = 1;
              }
          }

        if (specifier && specifier_width)
          {
            for (w = 0; format[f] >= '0' && format[f] <= '9'
                 && w < str_size; w++)
              {
                width_string[w] = format[f];
                f++;
              }
            width_string[w] = '\0';
```

```
            specifier_width = 0;

            if (format[f] == '.')
              {
                specifier_precision = 1;
                f++;
              }
            else
              {
                specifier_precision = 0;
                specifier_type      = 1;
              }
          }

        if (specifier && specifier_precision)
          {
            for (p = 0; format[f] >= '0' && format[f] <= '9'
                 && p < str_size; p++)
              {
                precision_string[p] = format[f];
                p++;
              }
            precision_string[p] = '\0';

            specifier_precision = 0;
            specifier_type      = 1;
          }

        if (specifier && specifier_type)
          {
            width     = atoi (width_string);
            precision = atoi (precision_string);
                              filler = ' ';
            if (flag_zero)    filler = '0';
            if (flag_space)   filler = ' ';
                              alignment = width;
            if (flag_minus)
              {
                alignment = -alignment;
                filler = ' ';    // Il carattere di riempimento
                                 // non può essere zero.
              }

            if (format[f] == 'h' && format[f+1] == 'h')
              {
                if (format[f+2] == 'd' || format[f+2] == 'i')
                  {
                    // signed char, base 10.
                    value_i = va_arg (ap, int);
```

```
                if (flag_plus)
                  {
                    s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                        alignment, filler, remain);
                  }
                else
                  {
                    s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                       alignment, filler, remain);
                  }
                f += 3;
              }
            else if (format[f+2] == 'u')
              {
                // unsigned char, base 10.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                    alignment, filler, remain);
                f += 3;
              }
            else if (format[f+2] == 'o')
              {
                // unsigned char, base 8.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                    alignment, filler, remain);
                f += 3;
              }
            else if (format[f+2] == 'x')
              {
                // unsigned char, base 16.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                    alignment, filler, remain);
                f += 3;
              }
            else if (format[f+2] == 'X')
              {
                // unsigned char, base 16.
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                    alignment, filler, remain);
                f += 3;
              }
            else if (format[f+2] == 'b')
              {
                // unsigned char, base 2 (estensione).
                value_ui = va_arg (ap, unsigned int);
                s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
```

```
                                  alignment, filler, remain);
              f += 3;
            }
          else // Specificatore errato;
            {
              f += 2;
            }
        }
      else if (format[f] == 'h')
        {
          if (format[f+1] == 'd' || format[f+1] == 'i')
            {
              // short int, base 10.
              value_i = va_arg (ap, int);
              if (flag_plus)
                {
                  s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                      alignment, filler, remain);
                }
              else
                {
                  s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                     alignment, filler, remain);
                }
              f += 2;
            }
          else if (format[f+1] == 'u')
            {
              // unsigned short int, base 10.
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'o')
            {
              // unsigned short int, base 8.
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'x')
            {
              // unsigned short int, base 16.
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                  alignment, filler, remain);
              f += 2;
```

```
            }
          else if (format[f+1] == 'X')
            {
              // unsigned short int, base 16.
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'b')
            {
              // unsigned short int, base 2 (estensione).
              value_ui = va_arg (ap, unsigned int);
              s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else // Specificatore errato;
            {
              f += 1;
            }
        }
      //
      // Il tipo «long long int» non c'è, perché il compilatore
      // GNU C, per poter eseguire le divisioni e il calcolo del
      // resto, ha bisogno delle funzioni di libreria
      // «__udivdi3()» e «__umoddi3()».
      //
      else if (format[f] == 'l')
        {
          if (format[f+1] == 'd' || format[f+1] == 'i')
            {
              // long int base 10.
              value_i = va_arg (ap, long int);
              if (flag_plus)
                {
                  s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                      alignment, filler, remain);
                }
              else
                {
                  s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                     alignment, filler, remain);
                }
              f += 2;
            }
          else if (format[f+1] == 'u')
            {
              // Unsigned long int base 10.
```

```
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'o')
              {
                // Unsigned long int base 8.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'x')
              {
                // Unsigned long int base 16.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'X')
              {
                // Unsigned long int base 16.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'b')
              {
                // Unsigned long int base 2 (estensione).
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else // Specificatore errato;
              {
                f += 1;
              }
          }
        else if (format[f] == 'j')
          {
            if (format[f+1] == 'd' || format[f+1] == 'i')
              {
                // intmax_t base 10.
                value_i = va_arg (ap, intmax_t);
                if (flag_plus)
```

```
                {
                  s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                      alignment, filler, remain);
                }
              else
                {
                  s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                     alignment, filler, remain);
                }
              f += 2;
            }
          else if (format[f+1] == 'u')
            {
              // uintmax_t base 10.
              value_ui = va_arg (ap, uintmax_t);
              s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'o')
            {
              // uintmax_t base 8.
              value_ui = va_arg (ap, uintmax_t);
              s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'x')
            {
              // uintmax_t base 16.
              value_ui = va_arg (ap, uintmax_t);
              s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'X')
            {
              // uintmax_t base 16.
              value_ui = va_arg (ap, uintmax_t);
              s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'b')
            {
              // uintmax_t base 2 (estensione).
              value_ui = va_arg (ap, uintmax_t);
              s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                  alignment, filler, remain);
```

```
              f += 2;
            }
          else // Specificatore errato;
            {
              f += 1;
            }
        }
      else if (format[f] == 'z')
        {
          if (format[f+1] == 'd'
              || format[f+1] == 'i'
              || format[f+1] == 'i')
            {
              // size_t base 10.
              value_ui = va_arg (ap, unsigned long int);
              s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'o')
            {
              // size_t base 8.
              value_ui = va_arg (ap, unsigned long int);
              s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'x')
            {
              // size_t base 16.
              value_ui = va_arg (ap, unsigned long int);
              s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'X')
            {
              // size_t base 16.
              value_ui = va_arg (ap, unsigned long int);
              s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                  alignment, filler, remain);
              f += 2;
            }
          else if (format[f+1] == 'b')
            {
              // size_t base 2 (estensione).
              value_ui = va_arg (ap, unsigned long int);
              s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                  alignment, filler, remain);
```

```
                f += 2;
              }
            else // Specificatore errato;
              {
                f += 1;
              }
          }
        else if (format[f] == 't')
          {
            if (format[f+1] == 'd' || format[f+1] == 'i')
              {
                // ptrdiff_t base 10.
                value_i = va_arg (ap, long int);
                if (flag_plus)
                  {
                    s += simaxtoa_fill (value_i, &string[s], 10, 0,
                                        alignment, filler, remain);
                  }
                else
                  {
                    s += imaxtoa_fill (value_i, &string[s], 10, 0,
                                       alignment, filler, remain);
                  }
                f += 2;
              }
            else if (format[f+1] == 'u')
              {
                // ptrdiff_t base 10, senza segno.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'o')
              {
                // ptrdiff_t base 8, senza segno.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                    alignment, filler, remain);
                f += 2;
              }
            else if (format[f+1] == 'x')
              {
                // ptrdiff_t base 16, senza segno.
                value_ui = va_arg (ap, unsigned long int);
                s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                    alignment, filler, remain);
                f += 2;
              }
```

```
    else if (format[f+1] == 'X')
      {
        // ptrdiff_t base 16, senza segno.
        value_ui = va_arg (ap, unsigned long int);
        s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                            alignment, filler, remain);
        f += 2;
      }
    else if (format[f+1] == 'b')
      {
        // ptrdiff_t base 2, senza segno.
        value_ui = va_arg (ap, unsigned long int);
        s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                            alignment, filler, remain);
        f += 2;
      }
    else // Specificatore errato;
      {
        f += 1;
      }
  }
if (format[f] == 'd' || format[f] == 'i')
  {
    // int base 10.
    value_i = va_arg (ap, int);
    if (flag_plus)
      {
        s += simaxtoa_fill (value_i, &string[s], 10, 0,
                            alignment, filler, remain);
      }
    else
      {
        s += imaxtoa_fill (value_i, &string[s], 10, 0,
                           alignment, filler, remain);
      }
    f += 1;
  }
else if (format[f] == 'u')
  {
    // unsigned int base 10.
    value_ui = va_arg (ap, unsigned int);
    s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
                        alignment, filler, remain);
    f += 1;
  }
else if (format[f] == 'o')
  {
    // Unsigned int base 8.
    value_ui = va_arg (ap, unsigned int);
```

```
            s += uimaxtoa_fill (value_ui, &string[s], 8, 0,
                                alignment, filler, remain);
            f += 1;
          }
        else if (format[f] == 'x')
          {
            // unsigned int base 16.
            value_ui = va_arg (ap, unsigned int);
            s += uimaxtoa_fill (value_ui, &string[s], 16, 0,
                                alignment, filler, remain);
            f += 1;
          }
        else if (format[f] == 'X')
          {
            // unsigned int base 16.
            value_ui = va_arg (ap, unsigned int);
            s += uimaxtoa_fill (value_ui, &string[s], 16, 1,
                                alignment, filler, remain);
            f += 1;
          }
        else if (format[f] == 'b')
          {
            // unsigned int base 2 (estensione).
            value_ui = va_arg (ap, unsigned int);
            s += uimaxtoa_fill (value_ui, &string[s], 2, 0,
                                alignment, filler, remain);
            f += 1;
          }
        //else if (format[f] == 'c')
        //  {
        //    // unsigned char.
        //    value_ui = va_arg (ap, unsigned int);
        //    s += uimaxtoa_fill (value_ui, &string[s], 10, 0,
        //                        alignment, filler, remain);
        //    f += 1;
        //  }
        else if (format[f] == 'c')
          {
            // unsigned char.
            value_ui = va_arg (ap, unsigned int);
            string[s] = (char) value_ui;
            s += 1;
            f += 1;
          }
        else if (format[f] == 's')
          {
            // string.
            value_cp = va_arg (ap, char *);
            filler = ' ';
```

```
                s += strtostr_fill (value_cp, &string[s], alignment,
                                    filler, remain);
                f += 1;
              }
            else // Specificatore errato;
              {
                ;
              }
            //
            // Fine dello specificatore.
            //
            width_string[0]     = '\0';
            precision_string[0] = '\0';

            specifier           = 0;
            specifier_flags     = 0;
            specifier_width     = 0;
            specifier_precision = 0;
            specifier_type      = 0;

            flag_plus           = 0;
            flag_minus          = 0;
            flag_space          = 0;
            flag_alternate      = 0;
            flag_zero           = 0;
          }
      }
    string[s] = '\0';
    return s;
}
```

Listato u9.32. '`./05/lib/snprintf.c`'

```
#include <stdio.h>
int
snprintf (char *restrict string, size_t n, const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vsnprintf (string, n, format, ap);
}
```

Librerie specifiche generali

Dopo le librerie standard vanno predisposte anche altre librerie specifiche per il proprio sistema. Quelle descritte nelle sezioni successive sono quelle di uso generale.

File «build.h»

Il file '`05/include/kernel/build.h`' viene prodotto dallo script '`05/makeit`', allo scopo di generare la macro-variabile ***BUILD_DATE*** contenente il momento esatto della compilazione. Durante gli esperimenti per la realizzazione del sistema è importante rendersi conto se ciò che si sta osservando corrisponde effettivamente al risultato dell'ultima compilazione oppure no. Il contenuto del file ha un aspetto simile a quello seguente:

```
#define BUILD_DATE "20070817191030"
```

Libreria «io.h»

La libreria rappresentata dal file di intestazione '`io.h`' contiene la dichiarazione di funzioni necessarie alla comunicazione con le componenti hardware. In questo caso si utilizzano solo funzioni per riprodurre le istruzioni '**`INB`**' e '**`OUTB`**' del linguaggio assemblatore, ma potrebbe essere estesa anche con altre funzioni per istruzioni analoghe, per la comunicazione con dati di dimensione maggiore del byte.

Listato u9.34. '`./05/include/kernel/io.h`'

```
#ifndef _IO_H
#define _IO_H   1

void         outb (unsigned int port, unsigned int data);
unsigned int inb  (unsigned int port);

#endif
```

Naturalmente è necessario realizzare entrambe le funzioni. È il caso di ricordare che il valore restituito dalle funzioni scritte in linguaggio assemblatore è quello contenuto nel registro ***EAX***.

Listato u9.35. '`./05/lib/io/inb.s`'

```
.globl  inb
#
inb:
    enter $4, $0
    pusha
    .equ inb_port,  8            # Primo parametro.
    .equ inb_data, -4            # Variabile locale.
    mov  inb_port(%ebp), %edx    # Successivamente si usa
                                 # solo DX.
    inb %dx, %al
    mov  %eax, inb_data(%ebp)    # Salva EAX nella variabile
                                 # locale.
    popa
    mov  inb_data(%ebp), %eax    # Recupera EAX e termina.
    leave
    ret
```

Listato u9.36. '`./05/lib/io/outb.s`'

```
.globl  outb
#
outb:
    enter $0, $0
    pusha
    .equ outb_port,  8           # Primo parametro.
    .equ outb_data, 12           # Secondo parametro.
    mov  outb_port(%ebp), %edx   # Successivamente si usa
                                 # solo DX.
    mov  outb_data(%ebp), %eax   # Successivamente si usa
                                 # solo AL.
    outb %al, %dx
    popa
    leave
    ret
```

La libreria rappresentata dal file di intestazione '`multiboot.h`' contiene semplicemente una struttura per facilitare la lettura delle informazioni più importanti che offre un sistema di avvio aderente alle specifiche *multiboot*; inoltre dichiara due funzioni: una per la raccolta delle informazioni e l'altra per la loro visualizzazione.

Listato u9.37. '`./05/include/kernel/multiboot.h`'

```
#ifndef _MULTIBOOT_H
#define _MULTIBOOT_H    1

#include <inttypes.h>

typedef struct {
    uint32_t  flags;
    uint32_t  mem_lower;
    uint32_t  mem_upper;
    uint32_t  boot_device;
        char *cmdline;
} multiboot_t;

void mboot_info (multiboot_t *info);
void mboot_show (void);

#endif
```

La funzione ***mboot_info()*** deve raccogliere e salvare le informazioni *multiboot*, all'interno della variabile strutturata ***os.multiboot*** (la variabile ***os*** complessiva è descritta nel file '`os.h`').

Listato u9.38. '`./05/lib/multiboot/mboot_info.c`'

```
#include <kernel/multiboot.h>
#include <string.h>
#include <stdio.h>
void
mboot_info (multiboot_t *info)
{
    os.multiboot.flags = info->flags;
    //
    if ((info->flags & 1) > 0)
      {
        os.multiboot.mem_lower = info->mem_lower;
```

```
        os.multiboot.mem_upper = info->mem_upper;
      }
    if ((info->flags & 2) > 0)
      {
        os.multiboot.boot_device = info->boot_device;
      }
    if ((info->flags & 4) > 0)
      {
        strncpy (os.multiboot.cmdline, info->cmdline, 1024);
      }
}
```

La funzione ***mboot_show()*** deve visualizzare direttamente le informazioni *multiboot*, salvate in precedenza, pertanto si avvale della funzione ***printf()*** che deve essere ancora descritta.

Listato u9.39. '`./05/lib/multiboot/mboot_show.c`'

```
#include <kernel/multiboot.h>
#include <stdio.h>
void
mboot_show (void)
{
    printf ("[%s] flags: %032b ", __func__,
            os.multiboot.flags);
    //
    if ((os.multiboot.flags & 1) > 0)
      {
        printf ("mlow: %04X mhigh: %08X",
                os.multiboot.mem_lower,
                os.multiboot.mem_upper);
      }
    printf ("\n");
    printf ("[%s] ", __func__);
    if ((os.multiboot.flags & 2) > 0)
      {
        printf ("bootdev: %08X ", os.multiboot.boot_device);
      }
    if ((os.multiboot.flags & 4) > 0)
      {
        printf ("cmdline: \"%s\"", os.multiboot.cmdline);
      }
```

```
    printf ("\n");
}
```

File «os.h»

Il file di intestazione '`os.h`' serve esclusivamente per definire una struttura, con la quale si crea la variabile strutturata ***os***, accessibile a ogni parte del sistema. In questa superstruttura vengono annotate tutte le informazioni che devono essere condivise. Il senso delle varie componenti della variabile ***os*** si chiarisce successivamente; a ogni modo è importante osservare che nel sistema non vengono usate altre variabili pubbliche.

Listato u9.40. '`./05/include/kernel/os.h`'

```
#ifndef _OS_H
#define _OS_H   1

#include <stdint.h>
#include <kernel/multiboot.h>
#include <stdbool.h>
#include <time.h>

typedef struct {
    //
    // Multiboot.
    //
    struct {
        uint32_t  flags;
        uint32_t  mem_lower;
        uint32_t  mem_upper;
        uint32_t  boot_device;
        char      cmdline[1024];
    } multiboot;
    //
    // Stato dello schermo VGA.
    //
    struct {
        unsigned short *video;
        unsigned short  columns;
        unsigned short  rows;
```

```
        unsigned int    position;
        unsigned char   attribute;
    } vga;
    //
    // «os.mem_ph»    Mappa della memoria fisica.
    //
    struct {
        uintptr_t  total_s;     // «..._s» = start
        uintptr_t  total_e;     // «..._e» = end.
           size_t  total_l;     // «..._l» = limit.
        uintptr_t  k_text_s;    // «k_...» = kernel.
        uintptr_t  k_text_e;    //
        uintptr_t  k_rodata_s;  //
        uintptr_t  k_rodata_e;  //
        uintptr_t  k_data_s;    //
        uintptr_t  k_data_e;    //
        uintptr_t  k_bss_s;     //
        uintptr_t  k_bss_e;     //
        uintptr_t  available_s; //
        uintptr_t  available_e; //
    } mem_ph;
    //
    // «os.gtd»                Tabella GTD.
    //
    union {
        struct {
            uint32_t  limit_a                 : 16,
                      base_a                  : 16;
            uint32_t  base_b                  : 8,
                      accessed                : 1,
                      write_execute           : 1,
                      expansion_conforming : 1,
                      code_or_data            : 1,
                      code_data_or_system  : 1,
                      dpl                     : 2,
                      present                 : 1,
                      limit_b                 : 4,
                      available               : 1,
```

```
                    reserved               : 1,
                    big                    : 1,
                    granularity            : 1,
                    base_c                 : 8;
        } cd;
        struct {
            uint32_t  limit_a              : 16,
                      base_a               : 16;
            uint32_t  base_b               : 8,
                      type                 : 4,
                      code_data_or_system  : 1,
                      dpl                  : 2,
                      present              : 1,
                      limit_b              : 4,
                      reserved             : 3,
                      granularity          : 1,
                      base_c               : 8;
        } system;
    } gdt[3];
    //
    // «os.gtdr»      Registro GTDR.
    //
    // È necessario che la struttura sia compattata, in modo
    // da usare complessivamente 48 bit; pertanto si usa
    // l'attributo «packed» del compilatore GNU C.
    //
    struct {
        uint16_t  limit;
        uint32_t  base;
    } __attribute__ ((packed)) gdtr;
    //
    // «os.idt»               Tabella IDT.
    //
    struct {
        uint32_t  offset_a : 16,
                  selector : 16;
        uint32_t  filler   : 8,
                  type     : 4,
```

```
                 system   : 1,
                 dpl      : 2,
                 present  : 1,
                 offset_b : 16;
    } idt[129];
    //
    // «os.idtr»      Registro IDTR.
    //
    // È necessario che la struttura sia compattata, in modo
    // da usare complessivamente 48 bit; pertanto si usa
    // l'attributo «packed» del compilatore GNU C.
    //
    struct {
        uint16_t  limit;
        uint32_t  base;
    } __attribute__ ((packed)) idtr;
    //
    // PIT: programmable interval timer.
    //
    struct {
        clock_t freq;
        clock_t clocks;
    } timer;
    //
    // Stato della tastiera.
    //
    struct {
        bool shift;
        bool shift_lock;
        bool ctrl;
        bool alt;
        bool echo;
        char key;
        char map1[128];
        char map2[128];
    } kbd;
    //
} os_t;
```

```
//
// Struttura pubblica con tutte le informazioni sul sistema.
//
os_t os;

#endif
```

Libreria «vga.h»

La libreria che fa capo al file di intestazione 'vga.h' è responsabile della visualizzazione del testo attraverso lo schermo.

Listato u9.41. './05/include/kernel/vga.h'

```
#ifndef _VGA_H
#define _VGA_H 1

#include <restrict.h>
#include <kernel/io.h>
#include <kernel/os.h>
#include <stddef.h>
#include <stdarg.h>
#include <stdint.h>
#include <stdio.h>

#define vga_char(c, attrib) \
    ((int16_t) c | ((((int16_t) attrib) << 8) & 0xFF00))

void vga_init     (void);
void vga_set      (unsigned short *video, int columns,
                   int rows, int x, int y, int position,
                   int attribute);
int  vga_clear    (void);
void vga_new_line (void);
void vga_putc     (int c);
void vga_puts     (char *string, size_t n);
int  vga_vprintf  (const char *restrict format,
                   va_list arg);
int  vga_printf   (const char *restrict format, ...);
```

```
#define clear()    (vga_clear ())
#define echo()     (os.kbd.echo = 1)
#define noecho()   (os.kbd.echo = 0)

#endif
```

Alcune macro-istruzioni definite nel file '`vga.h`' si limitano a scrivere un valore all'interno di ***os.kbd.echo***, la quale, se attiva, rappresenta la richiesta di visualizzare sullo schermo il testo che viene digitato. La macro-istruzione ***vga_char()*** assembla due valori in modo da ottenere un valore a 16 bit adatto alla visualizzazione sullo schermo di un carattere (l'unione dell'attributo di visualizzazione e del carattere stesso).

La funzione '`vga_init()`' va usata prima di fare qualunque cosa con lo schermo VGA, per attribuire dei valori iniziali corretti alla struttura ***os.vga***, la quale serve a tenere memoria della posizione corrente del cursore di scrittura e dell'attributo corrente da usare per i colori dei caratteri da scrivere.

Listato u9.42. '`./05/lib/vga/vga_init.c`'

```
#include <kernel/vga.h>
void
vga_init (void)
{
    os.vga.video     = (unsigned short *) 0xB8000;
    os.vga.columns   = 80;
    os.vga.rows      = 25;
    os.vga.position  = 0;
    os.vga.attribute = 0x07;
}
```

La funzione '`vga_set()`' che appare nel listato successivo ha lo scopo di spostare e di tenere traccia della posizione corrente del cursore di scrittura, in base alle informazioni che gli vengono fornite, determinando il resto in modo predefinito. Va osservato che, quando si tratta di valori interi, per dire alla funzione ***vga_set()*** di utilizzare i dati predefiniti si trasmette un valore negativo.

Listato u9.43. '`./05/lib/vga/vga_set.c`'

```
#include <kernel/vga.h>
void
vga_set (unsigned short *video,
         int columns, int rows,
         int x, int y,
         int position,
```

```
         int attribute)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned int screen_size = os.vga.columns * os.vga.rows;
    char position_high;
    char position_low;
    //
    if (video != NULL)  os.vga.video   = video;
    if (columns >= 0)   os.vga.columns = columns;
    if (rows >= 0)      os.vga.rows    = rows;
    if (columns >= 0 || rows >= 0)
                        screen_size    = os.vga.columns * os.vga.rows;
    if (x >= 0)         current_x      = x;
    if (y >= 0)         current_y      = y;
    if (x >= 0 || y >= 0)
      {
        os.vga.position   = current_y * os.vga.columns + current_x;
        os.vga.position   = os.vga.position % screen_size;
      }
    if (position >= 0)
      {
        //
        // Ricalcola la posizione anche se è già stata determinata
        // con i parametri "x" and "y".
        //
        os.vga.position   = position % screen_size;
      }
    if (x >= 0 || y >= 0 || position >= 0)
      {
        //
        // Deve riposizionare il cursore.
        //
        position_high = (unsigned char) (os.vga.position >> 8);
        position_low  = (unsigned char) os.vga.position;
        //
        outb (0x3D4, 0x0E);
        outb (0x3D5, position_high);
        outb (0x3D4, 0x0F);
        outb (0x3D5, position_low);
      }
    if (attribute >= 0) os.vga.attribute = attribute;
}
```

Listato u9.44. ‘./05/lib/vga/vga_clear.c’

```
#include <kernel/vga.h>
int
vga_clear (void)
{
    unsigned short blank = vga_char (' ', os.vga.attribute);
    unsigned int   i;
    unsigned int   screen_size = os.vga.columns * os.vga.rows;

    for (i = 0; i < screen_size ; i++)
      {
        *(os.vga.video + i) = blank;
      }
    return 0;   // Per essere compatibile, in qualche modo, con «clear()».
}
```

Listato u9.45. ‘./05/lib/vga/vga_new_line.c’

```
#include <kernel/vga.h>
void
vga_new_line (void)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned short blank       = vga_char (' ', os.vga.attribute);
    unsigned int   screen_size = os.vga.columns * os.vga.rows;
    int i;
    int j;

    current_x = 0;
    current_y++;

    if (current_y >= os.vga.rows)
      {
        //
        // Copia il testo in su di una riga.
        //
        for (i = 0, j = os.vga.columns; j < screen_size; i++, j++)
          {
            *(os.vga.video + i) = *(os.vga.video + j);
          }
        //
        // Ripulisce l'ultima riga di testo.
        //
        for (i = screen_size - os.vga.columns; i < screen_size; i++)
          {
            *(os.vga.video + i) = blank;
          }
        current_y--;
```

```
      }
    vga_set (NULL, -1, -1, current_x, current_y, -1, -1);
}
```

Listato u9.46. ‘./05/lib/vga/vga_putc.c’

```
#include <kernel/vga.h>
void
vga_putc (int c)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned short int cell;

    if (c == '\n' || c == '\r')
      {
        vga_new_line ();
      }
    else
      {
        cell = vga_char (c, os.vga.attribute);

        *(os.vga.video + os.vga.position) = cell;

        if (current_x == os.vga.columns)
          {
            vga_new_line ();
          }
        else
          {
            vga_set (NULL, -1, -1, -1, -1, os.vga.position + 1, -1);
          }
      }
}
```

Listato u9.47. '`./05/lib/vga/vga_puts.c`'

```
#include <kernel/vga.h>
void
vga_puts (char *string, size_t n)
{
    size_t i;
    for (i = 0; i < n ; i++)
      {
        if (string[i] == 0) break;
        if (string[i] != 0) vga_putc (string[i]);
      }
    // Non aggiunge "\n"!
}
```

Listato u9.48. '`./05/lib/vga/vga_vprintf.c`'

```
#include <kernel/vga.h>
int
vga_vprintf (const char *restrict format, va_list arg)
{
    const size_t dim = 2000;  // Dimensione massima dello schermo: 25x80.
    char  string[dim];
    int   ret;
    string[0] = 0;
    ret = vsprintf(string, format, arg);
    vga_puts (string, dim);
    return ret;
}
```

Listato u9.49. '`./05/lib/vga/vga_printf.c`'

```
#include <kernel/vga.h>
int
vga_printf  (const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vga_vprintf (format, ap);
}
```

Un primo kernel di prova

Avviando il sistema con GRUB 1 o con un altro programma conforme alle specifiche *multiboot*, il kernel dovrebbe trovarsi già in un contesto funzionante in modalità protetta, utilizzando tutta la memoria in modo lineare (ovvero senza suddivisione in segmenti). Pertanto, per visualizzare qualcosa sullo schermo non è indispensabile il passare subito alla preparazione della tabella GDT, cosa che consente di verificare se i file già preparati sono corretti.

In queste sezioni vengono descritti altri file del sistema in fase di sviluppo, ma in particolare '`kernel_main.c`' non è ancora nella sua impostazione definitiva, per consentire una verifica provvisoria del lavoro.

File «kernel.h»

Il file di intestazione '`kernel.h`' viene usato soprattutto per definire le funzioni principali del kernel, ma si possono notare, in coda, delle funzioni che in realtà non esistono, corrispondenti a simboli generati attraverso il «collegatore» (il *linker*). Queste funzioni fantasma servono solo per consentire l'individuazione degli indirizzi rispettivi, così da sapere come è disposto in memoria il kernel.

Listato u9.50. '`./05/include/kernel/kernel.h`'

```
#ifndef _KERNEL_H
#define _KERNEL_H       1

#include <restrict.h>
#include <kernel/multiboot.h>
#include <kernel/os.h>
//
// Funzioni principali da cui inizia l'esecuzione del kernel.
//
void kernel_boot        (void);
void kernel_main        (unsigned long magic, multiboot_t *info);
void kernel_memory      (multiboot_t *info);
void kernel_memory_show (void);
//
// Simboli di riferimento inseriti dallo script di LD (linker script).
// Vengono dichiarate qui come funzioni, solo per comodità, ma servono
// solo per individuare le posizioni utilizzate dal kernel nella memoria
// fisica, così da poter costruire poi una tabella GDT decente.
//
void k_mem_total_s  (void);
void k_mem_text_s   (void);
void k_mem_text_e   (void);
```

```
void k_mem_rodata_s (void);
void k_mem_rodata_e (void);
void k_mem_data_s   (void);
void k_mem_data_e   (void);
void k_mem_bss_s    (void);
void k_mem_bss_e    (void);
void k_mem_total_e  (void);

#endif
```

La funzione ***kernel_boot()*** è quella responsabile dell'avvio ed è scritta necessariamente in linguaggio assemblatore. Si trova contenuta nel file '`kernel_boot.s`', assieme alla dichiarazione dell'impronta di riconoscimento *multiboot* e alla collocazione dello spazio usato per la pila dei dati (l'unica pila che questo piccolo sistema utilizzi). È attraverso la configurazione del collegatore, nel file '`linker.ld`', che viene specificato di partire con la funzione ***kernel_boot()***.

Listato u9.51. '`./05/kernel/kernel_boot.s`'

```
.extern kernel_main
#
.globl  kernel_boot
#
# Dimensione della pila interna al kernel. Qui vengono previsti
# 32768 byte (0x8000 byte).
#
.equ STACK_SIZE,  0x8000
#
# Si inizia subito con il codice che si mescola con i dati;
# pertanto si deve saltare alla procedura che deve predisporre
# la pila e avviare il kernel scritto in C.
#
kernel_boot:
    jmp start
#
# Per collocare correttamente i dati che si trovano dopo l'istruzione
# di salto, si fa in modo di riempire lo spazio mancante al
# completamento di un blocco di 4 byte.
#
.align 4
#
# Intestazione «multiboot» che deve apparire poco dopo l'inizio
# del file-immagine.
#
multiboot_header:
    .int 0x1BADB002                    # magic
```

```
    .int 0x00000003                    # flags
    .int -(0x1BADB002 + 0x00000003)    # checksum
#
# Inizia il codice di avvio.
#
start:
    #
    # Regola ESP alla base della pila.
    #
    movl $(stack_max + STACK_SIZE), %esp
    #
    # Azzera gli indicatori contenuti in EFLAGS, ma per questo deve
    # usare la pila appena sistemata.
    #
    pushl $0
    popf
    #
    # Chiama la funzione principale scritta in C, passandogli le
    # informazioni ottenute dal sistema di avvio.
    #
    # void kernel_main (unsigned int magic, void *multiboot_info)
    #
    pushl %ebx          # Puntatore alla struttura contenente le
                        # informazioni passate dal sistema di avvio.
    pushl %eax          # Codice di riconoscimento del sistema di avvio.
    #
    call kernel_main    # Chiama la funzione kernel().
#
# Procedura di arresto.
#
halt:
    hlt                 # Se il kernel termina, ferma il microprocessore.
    jmp halt            # Se il microprocessore viene sbloccato, si
                        # ripete il comando HLT.
#
# Alla fine viene collocato lo spazio per la pila dei dati,
# senza inizializzarlo. Per scrupolo si allinea ai 4 byte (32 bit).
#
.align 4
.comm stack_max, STACK_SIZE
```

La funzione ***kernel_main()*** (avviata da ***kernel_boot()***) che viene mostrata nel listato successivo, non è ancora nella sua forma definitiva: per il momento si limita alla visualizzazione delle informazioni ***multiboot*** e allo stato della memoria utilizzata.

Listato u9.52. Prima versione del file '`./05/kernel/kernel_main.c`'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
void
kernel_main (unsigned long magic, multiboot_t *info)
{
    //
    // Inizializza i dati relativi alla gestione dello
    // schermo VGA, quindi ripulisce lo schermo.
    //
    vga_init ();
    clear ();
    //
    // Data e orario di compilazione.
    //
    printf ("05 %s\n", BUILD_DATE);
    //
    // Cerca le informazioni «multiboot».
    //
    if (magic == 0x2BADB002)
      {
        //
        // Salva e mostra le informazioni multiboot.
        //
        mboot_info (info);
        mboot_show ();
        //
        // Raccoglie i dati sulla memoria fisica.
        //
        kernel_memory (info);
        //
        // Omissis.
        //
      }
    else
      {
        printf ("[%s] no \"multiboot\" header!\n",
```

```
                 __func__);
      }
    //
    printf ("[%s] system halted\n", __func__);
    _Exit (0);
}
```

I listati successivi, relativi alle funzioni ***kernel_memory()*** e ***kernel_memory_show()***, sono nel loro stato definitivo.

Listato u9.53. '`./05/kernel/kernel_memory.c`'

```
#include <kernel/kernel.h>
#include <stdio.h>
void
kernel_memory (multiboot_t *info)
{
    //
    // Imposta valori conosciuti o predefiniti.
    //
    os.mem_ph.total_s     = (uint32_t) &k_mem_total_s;
    os.mem_ph.total_e     = (uint32_t) &k_mem_total_e;
    os.mem_ph.available_s = (uint32_t) &k_mem_total_e;
    os.mem_ph.available_e
     = (uint32_t) &k_mem_total_e+0x0FFFFF; // 1 Mibyte.
    //
    os.mem_ph.k_text_s   = (uint32_t) &k_mem_text_s;
    os.mem_ph.k_text_e   = (uint32_t) &k_mem_text_e;
    os.mem_ph.k_rodata_s = (uint32_t) &k_mem_rodata_s;
    os.mem_ph.k_rodata_e = (uint32_t) &k_mem_rodata_e;
    os.mem_ph.k_data_s   = (uint32_t) &k_mem_data_s;
    os.mem_ph.k_data_e   = (uint32_t) &k_mem_data_e;
    os.mem_ph.k_bss_s    = (uint32_t) &k_mem_bss_s;
    os.mem_ph.k_bss_e    = (uint32_t) &k_mem_bss_e;
    //
    if ((info->flags & 1) > 0)
      {
        os.mem_ph.available_e = 1024 * info->mem_upper;
      }
    //
```

```
    os.mem_ph.total_l = os.mem_ph.available_e / 0x1000;
    //
    kernel_memory_show ();
}
```

Listato u9.54. ‘./05/kernel/kernel_memory_show.c’

```
#include <kernel/kernel.h>
#include <stdio.h>
void
kernel_memory_show (void)
{
    //
    printf ("[%s] kernel %08" PRIX32 "..%08" PRIX32
            "  avail. %08" PRIX32 "..%08" PRIX32 "\n",
                    __func__,
                    os.mem_ph.total_s,
                    os.mem_ph.total_e,
                    os.mem_ph.available_s,
                    os.mem_ph.available_e);
    //
    printf ("[%s] text   %08" PRIX32 "..%08" PRIX32
            "  rodata %08" PRIX32 "..%08" PRIX32 "\n",
                    __func__,
                    os.mem_ph.k_text_s,
                    os.mem_ph.k_text_e,
                    os.mem_ph.k_rodata_s,
                    os.mem_ph.k_rodata_e);
    //
    printf ("[%s] data   %08" PRIX32 "..%08" PRIX32
            "  bss    %08" PRIX32 "..%08" PRIX32 "\n",
                    __func__,
                    os.mem_ph.k_data_s,
                    os.mem_ph.k_data_e,
                    os.mem_ph.k_bss_s,
                    os.mem_ph.k_bss_e);
    //
    printf ("[%s] limit  %08" PRIX32 "\n",
                    __func__,
```

```
                        os.mem_ph.total_l);
}
```

Altri file mancanti

Nella descrizione della libreria che fa capo al file di intestazione '`stdlib.h`', è stata omessa la funzione ***_Exit()*** che ora è indispensabile precisare, essendo usata dalla funzione ***kernel_main()***. In pratica si esegue semplicemente un ciclo senza fine, cercando però di sospendere il funzionamento del microprocessore, fino a quando si verifica un'interruzione.

Listato u9.55. '`./05/lib/_Exit.s`'

```
.globl _Exit
#
_Exit:
    enter $0, $0
    .equ status,  8         # Primo argomento.
    mov  status(%ebp), %eax # Copia il valore da restituire
                            # in EAX, anche se poi non se ne
                            # fa nulla.
    leave
    #
halt:
    hlt         # Ferma il microprocessore.
    jmp halt    # Se il microprocessore viene sbloccato, si
                # ripete il comando HLT.
```

Compilazione e prova di funzionamento

Prima di procedere alla compilazione con lo script '**`compile`**' (o direttamente con '**`makeit`**'), occorre verificare che la variabile di ambiente '**`TAB`**' sia dichiarata correttamente nello script '**`makeit`**', in modo da contenere esattamente un carattere di tabulazione orizzontale (diversamente i file-make non verrebbero creati nel modo giusto). Inoltre occorre avere preparato il file-immagine del dischetto e averlo innestato nella directory '`/mnt/fd0/`' (diversamente occorre modificare sempre lo script '**`makeit`**'). Quando tutto sembra pronto, basta avviare lo script '**`bochs`**' (da una finestra di terminale, durante una sessione grafica di lavoro con X) per far partire il sistema giocattolo in prova. Se tutto va bene, viene visualizzato il testo seguente e poi tutto si ferma; se invece si presenta un errore, il simulatore Bochs riavvia e si riparte con GRUB 1.

```
05 20070818140007
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010BAFC  avail. 0010BAFC..01EF0000
[kernel_memory_show] text   00100000..00102FEC  rodata 00102FEC..00103144
[kernel_memory_show] data   00103144..00103144  bss    00103160..0010BAFC
[kernel_memory_show] limit  00001EF0
[kernel_main] system halted
```

Dall'esempio mostrato si può determinare quanto segue: la memoria bassa arriva fino a $27F_{16}$ Kibyte (639 Kibyte); la memoria alta arriva fino a $7BC0_{16}$ Kibyte (31 680 Kibyte); il kernel utilizza la memoria da 100000_{16} byte (1 024 Kibyte) a $10BAFC_{16}$ byte (1 070 Kibyte circa); pertanto la parte rimanente è tutta memoria libera.

Con questi dati, nel prossimo gruppo di sezioni viene preparata una tabella GDT minima, con la quale si definisce solo la memoria esistente effettivamente.

Tabella GDT

La preparazione di una tabella GDT è indispensabile per poter accedere alla memoria in modalità protetta. Nel sistema in corso di realizzazione si intende usare la memoria in modo lineare, senza segmentazioni e senza pagine, pertanto si compila la tabella GDT con il minimo indispensabile, avendo cura di indicare in modo preciso la memoria esistente effettivamente.

Struttura

Nel file di intestazione '`os.h`' è già stata predisposta la struttura che facilita la compilazione e l'interpretazione dei descrittori della tabella GDT. In particolare viene usata un'unione, con due suddivisioni alternative: una per i descrittori di segmento codice o dati e l'altra per i descrittori di sistema. Per il lavoro in corso di realizzazione, i descrittori di sistema non vengono utilizzati, pertanto è sufficiente concentrarsi sulla struttura '**`os.gdt[`*n*`].cd`**':

```
...
    union {
        struct {
            uint32_t  limit_a              : 16,
                      base_a               : 16;
            uint32_t  base_b               : 8,
                      accessed             : 1,
                      write_execute        : 1,
                      expansion_conforming : 1,
```

```
                    code_or_data          : 1,
                    code_data_or_system   : 1,
                    dpl                   : 2,
                    present               : 1,
                    limit_b               : 4,
                    available             : 1,
                    reserved              : 1,
                    big                   : 1,
                    granularity           : 1,
                    base_c                : 8;
        } cd;
        struct {
            ...
            ...
        } system;
    } gdt[3];
...
```

Libreria «gdt.h»

Il file di intestazione 'gdt.h' contiene la dichiarazione delle funzioni che riguardano la gestione della tabella GDT.

Listato u9.58. './05/include/kernel/gdt.h'

```
#ifndef _GDT_H
#define _GDT_H  1

#include <inttypes.h>
#include <stdbool.h>
#include <kernel/os.h>

void gdt_desc_seg (int              descriptor,
                   uint32_t         base,
                   uint32_t         limit,
                   bool             present,
                   bool             granularity,
                   bool             code,
                   bool             write_execute,
                   bool             expand_down_non_conforming,
```

```
                        unsigned char dpl);
void gdt_print          (void *gdtr);
void gdt_load           (void *gdtr);
void gdt                (void);

#endif
```

La funzione ***gdt_desc_seg()*** serve a facilitare la compilazione di un descrittore della tabella; la funzione ***gdt_print()*** consente di visualizzare il contenuto della tabella, partendo dal contenuto del registro ***GDTR***, indipendentemente da altre informazioni; la funzione ***gdt_load()*** fa in modo che il microprocessore utilizzi il contenuto della tabella GDT; la funzione ***gdt()***, avvalendosi delle altre funzioni già citate, crea la tabella minima richiesta, ne mostra il contenuto e la attiva.

Listato u9.59. '`./05/lib/gdt/gdt_desc_seg.c`'

```
#include <kernel/gdt.h>
#include <stdio.h>
void
gdt_desc_seg (int           desc,
              uint32_t      base,
              uint32_t      limit,
              bool          present,
              bool          granularity,
              bool          code,
              bool          write_execute,
              bool          expand_down_non_conforming,
              unsigned char dpl)
{
    //
    // Verifica di non eccedere la dimensione dell'array.
    //
    int max = ((sizeof (os.gdt)) / 8) - 1;
    if (desc > max)
      {
        printf ("[%s] ERROR: selected descriptor %i when max is %i!\n",
                __func__, desc, max);
        return;
      }
    //
    // Limite.
    //
    os.gdt[desc].cd.limit_a = (limit & 0x0000FFFF);
    os.gdt[desc].cd.limit_b = limit / 0x10000;
```

```
    //
    // Indirizzo base.
    //
    os.gdt[desc].cd.base_a = (base & 0x0000FFFF);
    os.gdt[desc].cd.base_b = ((base / 0x10000) & 0x000000FF);
    os.gdt[desc].cd.base_c = (base / 0x1000000);
    //
    // Attributi.
    //
    os.gdt[desc].cd.accessed              = 0;
    os.gdt[desc].cd.write_execute         = write_execute;
    os.gdt[desc].cd.expansion_conforming = expand_down_non_conforming;
    os.gdt[desc].cd.code_or_data          = code;
    os.gdt[desc].cd.code_data_or_system  = 1;
    os.gdt[desc].cd.dpl                   = dpl;
    os.gdt[desc].cd.present               = present;
    os.gdt[desc].cd.available             = 0;
    os.gdt[desc].cd.reserved              = 0;
    os.gdt[desc].cd.big                   = 1;
    os.gdt[desc].cd.granularity           = granularity;
}
```

Listato u9.60. '`./05/lib/gdt/gdt_print.c`'

```
#include <kernel/gdt.h>
#include <stdio.h>
//
// Mostra il contenuto di una tabella GDT, a partire dal puntatore al
// registro GDTR in memoria. Pertanto non si avvale, volutamente, della
// struttura già predisposta con il linguaggio C, mentre «gdtr_t» viene
// creato qui solo provvisoriamente, per uso interno. Ciò serve ad
// assicurare che questa funzione compia il proprio lavoro in modo
// indipendente, garantendo la visualizzazione di dati reali.
//
typedef struct {
    uint16_t  limit;
    uint32_t  base;
} __attribute__ ((packed)) local_gdtr_t;
//
void
gdt_print (void *gdtr)
{
    local_gdtr_t *g = gdtr;
        uint32_t *p = (uint32_t *) g->base;
```

```
    int max = (g->limit + 1) / (sizeof (uint32_t));
    int i;

    printf ("[%s] base: 0x%08" PRIX32 " limit: 0x%04" PRIX32 "\n",
             __func__, g->base, g->limit);

    for (i = 0; i < max; i+=2)
      {
        printf ("[%s] %" PRIx32 " %032" PRIb32 " %032" PRIb32 "\n",
                __func__, i/2, p[i], p[i+1]);
      }
}
```

Listato u9.61. './05/lib/gdt/gdt_load.s'

```
.globl  gdt_load
#
gdt_load:
    enter $0, $0
    .equ gdtr_pointer,  8           # Primo argomento.
    mov  gdtr_pointer(%ebp), %eax   # Copia il puntatore
                                    # in EAX.
    leave
    #
    lgdt (%eax)    # Carica il registro GDTR dall'indirizzo
                   # in EAX.
    #
    # 2 dati per il kernel, DPL 0, comprendente tutta la
    #   memoria disponibile: selettore 0x10+0.
    #
    mov  $0x10, %ax
    mov  %ax,   %ds
    mov  %ax,   %es
    mov  %ax,   %fs
    mov  %ax,   %gs
    mov  %ax,   %ss
    #
    # 1 codice per il kernel, DPL 0, comprendente tutta
    #   la memoria disponibile: selettore 0x08+0.
```

```
    #
    jmp  $0x08, $flush
flush:
    ret
```

Listato u9.62. '`./05/lib/gdt/gdt.c`'

```
#include <kernel/gdt.h>
void
gdt (void)
{
    //
    // Imposta i dati necessari al registro GDTR.
    //
    os.gdtr.limit = (sizeof (os.gdt) - 1);
    os.gdtr.base  = (uint32_t) &os.gdt[0];
    //
    // Azzera le voci previste dell'array «os.gdt[]».
    // La prima di queste voci (0) rimane azzerata e non
    // deve essere utilizzata.
    //
    int i;
    for (i = 0; i < ((sizeof (os.gdt)) / 8); i++)
      {
        gdt_desc_seg (i, 0, 0, 0, 0, 0, 0, 0, 0);
      }
    //
    // 1 codice per il kernel, DPL 0, comprendente tutta la
    //   memoria disponibile: selettore 0x08+0.
    //
    gdt_desc_seg (1, 0,
                  os.mem_ph.total_l, 1, 1, 1, 1, 0, 0);
    //
    // 2 dati per il kernel, DPL 0, comprendente tutta la
    //   memoria disponibile: selettore 0x10+0.
    //
    gdt_desc_seg (2, 0,
                  os.mem_ph.total_l, 1, 1, 0, 1, 0, 0);
    //
```

```
    // Mostra la tabella GDT e poi la carica.
    //
    gdt_print (&os.gdtr);
    gdt_load  (&os.gdtr);
}
```

Modifiche da apportare a «kernel_main.c»

Nel file ‘`kernel_main.c`’ va aggiunta l’incorporazione del file ‘`gdt.h`’ e la chiamata alla funzione ***gdt()***:

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
...
        //
        // Raccoglie i dati sulla memoria fisica.
        //
        kernel_memory (info);
        //
        // Predispone la tabella GDT.
        //
        gdt ();
...
```

Una volta ricompilato il lavoro e avviato con Bochs, si deve ottenere una schermata simile a quella seguente:

```
05 20070819115151
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010BF7C  avail. 0010BF7C..01EF0000
[kernel_memory_show] text   00100000..00103418  rodata 00103418..001035FC
[kernel_memory_show] data   001035FC..001035FC  bss    00103600..0010BF7C
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010B648 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000010000000001000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001000000000
[kernel_main] system halted
```

Gestione della memoria

Nel sistema in corso di realizzazione non si intende gestire la memoria in modo sofisticato; in particolare non si vogliono usare né segmenti, né pagine. In pratica, lo spazio che rimane dopo l'intervallo usato dal kernel viene gestito con una lista e sulla base di questa impostazione vengono realizzate le funzioni '...`alloc()`' e *free()*.

Gestione della memoria attraverso una lista

Si intende gestire l'allocazione di memoria attraverso una lista in cui l'inizio di un blocco di memoria contenga il riferimento al blocco successivo e l'indicazione se il proprio sia un blocco libero o utilizzato; pertanto, come si vede nella figura, l'ultimo blocco punta al primo.

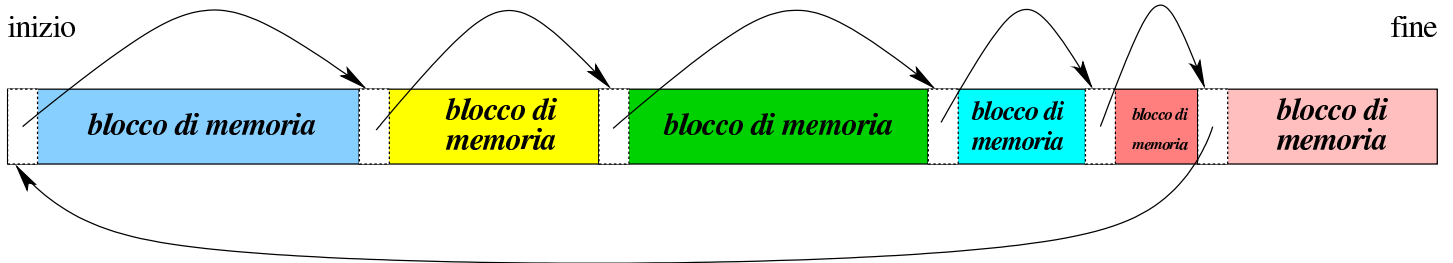
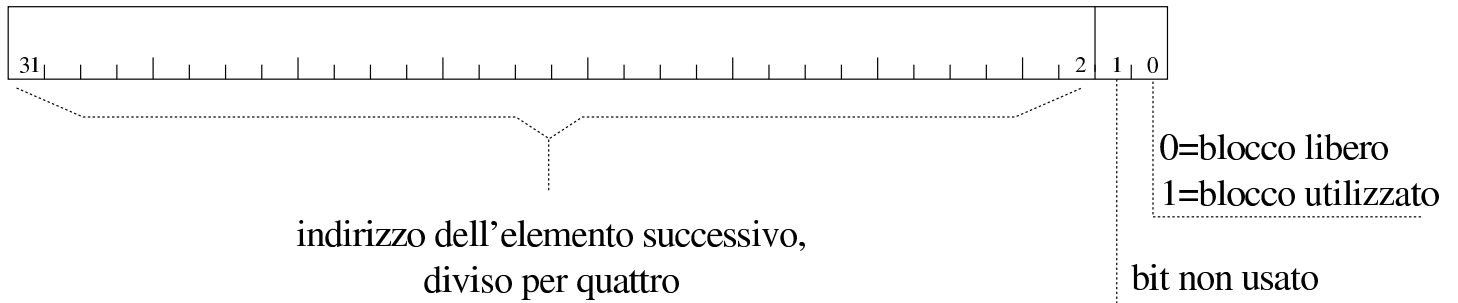
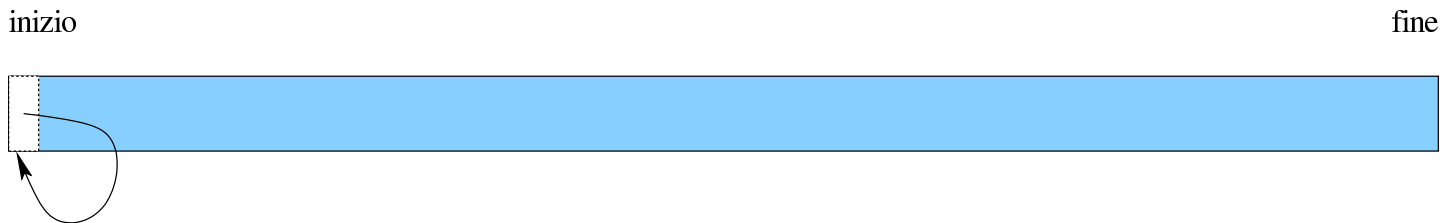
Figura u9.65. Lista di blocchi di memoria.

L'intestazione dei blocchi di memoria, con la quale si fa riferimento al blocco successivo e si annota se il blocco (proprio) è impegnato o meno, utilizza solo 32 bit, partendo dal presupposto che i blocchi di memoria debbano essere multipli di tale valore. A tale proposito si osservi che se i blocchi di memoria sono da 4 byte, gli indirizzi sono sempre multipli di quattro, ovvero di 100_2. Pertanto, i due bit meno significativi possono essere utilizzati per altri fini.

Figura u9.66. Struttura dell'intestazione dei blocchi di memoria.

Sulla base del principio affermato e di quanto si vede nella figura, l'indirizzo effettivo del blocco successivo si determina moltiplicando per quattro il valore annotato tra il bit 2 e il bit 31 dell'intestazione. Si osservi che l'indirizzo in questione è quello dell'inizio dell'intestazione del blocco successivo, pertanto il blocco di memoria successivo inizia effettivamente dopo altri quattro byte.

Quando la memoria viene inizializzata si crea un blocco solo, la cui intestazione punta a se stessa, come si vede nella figura successiva. Da questo si comprende anche che il blocco che punta a se stesso è lungo fino alla fine dello spazio di memoria disponibile complessivamente; inoltre si intende che con questo meccanismo, molto semplice, la memoria possa essere gestita solo se è presente in modo continuo. Questa semplificazione è stata fatta volutamente per non complicare inutilmente il codice; d'altra parte si osserva che così la «memoria bassa» (quella dei primi 640 Kibyte) non venga usata affatto.

Figura u9.67. La lista al momento iniziale.

inizio fine

Libreria «mm.h»

Il file di intestazione '`mm.h`' descrive la struttura usata per interpretare i primi 32 bit dei blocchi di memoria (per distinguere l'indirizzo successivo dall'indicazione dello stato del blocco attuale) e dichiara due funzioni per inizializzare la memoria e per leggerne la mappa.

Listato u9.68. '`./05/include/kernel/mm.h`'

```
#ifndef _MM_H
#define _MM_H 1

#include <restrict.h>
#include <stdint.h>
#include <inttypes.h>
#include <stddef.h>
#include <stdarg.h>
#include <kernel/os.h>

//
// La dimensione di «uintptr_t» condiziona la struttura
// «mm_head_t» e la dimensione delle unità minime di memoria
// allocata. «uintptr_t» è da 32 bit, così l'immagine del
// kernel è allineata a blocchi da 32 bit e così deve essere
// anche per gli altri blocchi di memoria. Essendo i blocchi
// di memoria multipli di 32 bit, gli indirizzi sono sempre
```

```
// multipli di 4 (4 byte); pertanto, servono solo 30 bit
// per rappresentare l'indirizzo, che poi viene ottenuto
// moltiplicandolo per quattro.
// Di conseguenza, il bit meno significativo viene usato
// per annotare se il blocco di memoria è libero e il bit
// successivo non viene usato. Questo meccanismo potrebbe
// essere usato anche con un indirizzamento a 16 bit, dove
// servirebbero 15 bit per indirizzi multipli di due byte.
//
typedef struct {
    uintptr_t allocated : 1,
              filler    : 1,
              next      : 30;
} mm_head_t;

void mm_init (void);
void mm_list (void);

#endif
```

La funzione ***mm_init()*** inizializza la memoria, creando un blocco libero che la descrive completamente. Per sapere dove inizia e dove finisce la memoria disponibile, si avvale delle informazioni contenute nella variabile strutturata ***os.mem_ph***, le quali sono state inserite precedentemente dalla funzione ***kernel_memory()***(listato u9.53)

Listato u9.69. '`./05/lib/mm/mm_init.c`'

```
#include <kernel/mm.h>
#include <stdio.h>
void
mm_init (void)
{
    uintptr_t  start = os.mem_ph.available_s;
    mm_head_t *head;
       size_t  available = os.mem_ph.available_e - os.mem_ph.available_s;
    //
    // La memoria disponibile deve essere di almeno 8 byte!
    //
    if (available < ((sizeof (mm_head_t)) * 2))
      {
        //
        // Il sistema viene fermato!
```

```
        //
        printf ("[%s] ERROR: not enough memory: %zu byte!\n",
                __func__, available);
        _Exit (0);
      }
    //
    // Predispone il nodo principale della lista.
    //
    head = (mm_head_t *) start;
    //
    // Inizializza il primo blocco, libero, che punta a se stesso,
    // essendo l'unico.
    //
    head->allocated = 0;
    head->next      = (start / (sizeof (mm_head_t)));
    //
    // Mostra come è andata.
    //
    printf ("[%s] available memory: %zu byte\n",
                __func__, available - (sizeof (mm_head_t)));
    //
    return;
}
```

La funzione ***mm_list()*** mostra la mappa della memoria gestita attraverso le funzioni '...**`alloc()`**'. Gli indirizzi che vengono forniti sono quelli di inizio dei blocchi, escludendo lo spazio utilizzato dalle intestazioni (pertanto, se l'intestazione inizia all'indirizzo ***n***, viene mostrato l'indirizzo ***n***+4).

Listato u9.70. '`./05/lib/mm/mm_list.c`'

```
#include <kernel/mm.h>
#include <stdio.h>
void mm_list (void)
{
    uintptr_t  start = os.mem_ph.available_s;
    mm_head_t *head  = (void *) start;
       size_t  actual_size;
    uintptr_t  current;
    uintptr_t  next;
    uintptr_t  up_to;
          int  counter;
    //
    // Scandisce la lista di blocchi di memoria.
    //
    counter = 2;
```

```
    while (counter)
      {
        //
        // Annota la posizione attuale e quella successiva.
        //
        current = (uintptr_t) head;
           next = head->next * (sizeof (mm_head_t));
        if (next == start)
          {
            up_to = os.mem_ph.available_e;
          }
        else
          {
            up_to = next;
          }
        //
        // Se è stato raggiunto il primo elemento, decrementa il
        // contatore di una unità. Se è già a zero, esce.
        //
        if (current == start)
          {
            counter--;
            if (counter == 0) break;
          }
        //
        // Determina la dimensione del blocco attuale.
        //
        if (current == start && next == start)
          {
            //
            // Si tratta del primo e unico elemento della lista.
            //
            actual_size = os.mem_ph.available_e - (sizeof (mm_head_t));
          }
        else
          {
            actual_size = up_to - current - (sizeof (mm_head_t));
          }
        //
        // Si mostra lo stato del blocco di memoria.
        //
        if (head->allocated)
          {
            printf ("[%s] used %08X..%08X size %08zX\n",
                    __func__,
```

```
                    current + (sizeof (mm_head_t)), up_to, actual_size);
          }
        else
          {
            printf ("[%s] free %08X..%08X size %08zX\n",
                    __func__,
                    current + (sizeof (mm_head_t)), up_to, actual_size);
          }
        //
        // Si passa alla posizione successiva.
        //
        head = (void *) next;
      }
}
```

Funzioni per l'allocazione della memoria

«

La funzione ***malloc()*** esegue una scansione della mappa della memoria, alla ricerca del primo blocco di dimensione sufficiente a soddisfare la richiesta ricevuta (*first fit*). Una volta trovato, se il blocco libero è abbastanza grande, lo divide, in modo da utilizzare solo lo spazio richiesto. Gli spazi allocati sono sempre multipli della dimensione di '**`mm_head_t`**', pertanto, se necessario, si alloca uno spazio leggermente più grande del richiesto.

Listato u9.71. '`./05/lib/malloc.c`'

```
#include <stdlib.h>
#include <kernel/mm.h>
void
*malloc (size_t size)
{
    uintptr_t  start = os.mem_ph.available_s;
    mm_head_t *head  = (void *) start;
       size_t  actual_size;
    uintptr_t  current;
    uintptr_t  next;
    uintptr_t  new;
    uintptr_t  up_to;
          int  counter;
    //
    // Arrotonda in eccesso il valore di «size», in modo che sia un
    // multiplo della dimensione di «mm_head_t». Altrimenti, la
    // collocazione dei blocchi successivi può avvenire in modo
    // non allineato.
    //
```

```
    size = (size + (sizeof (mm_head_t)) -1);
    size = size / (sizeof (mm_head_t));
    size = size * (sizeof (mm_head_t));
    //
    // Cerca un blocco libero di dimensione sufficiente.
    //
    counter = 2;
    while (counter)
      {
        //
        // Annota la posizione attuale e quella successiva.
        //
        current = (uintptr_t) head;
           next = head->next * (sizeof (mm_head_t));
        //
        if (next == start)
          {
            up_to = os.mem_ph.available_e;
          }
        else
          {
            up_to = next;
          }
        //
        // Se è stato raggiunto il primo elemento, decrementa il
        // contatore di una unità. Se è già a zero, esce.
        //
        if (current == start)
          {
            counter--;
            if (counter == 0) break;
          }
        //
        // Controlla se si tratta di un blocco libero.
        //

        if (! head->allocated)
          {
            //
            // Il blocco è libero: si deve determinarne la dimensione.
            //
            if (current == start && next == start)
              {
                //
                // Si tratta del primo e unico elemento della lista.
```

```
            //
            actual_size = os.mem_ph.available_e - (sizeof (mm_head_t));
          }
        else
          {
            actual_size = up_to - current - (sizeof (mm_head_t));
          }
        //
        // Si verifica che sia capiente.
        //
        if (actual_size >= size + ((sizeof (mm_head_t)) * 2))
          {
            //
            // C'è spazio per dividere il blocco.
            //
            new = current + size + (sizeof (mm_head_t));
            //
            // Aggiorna l'intestazione attuale.
            //
            head->allocated = 1;
            head->next = new / (sizeof (mm_head_t));
            //
            // Predispone l'intestazione successiva.
            //
            head = (void *) new;
            head->allocated = 0;
            head->next = next / (sizeof (mm_head_t));
            //
            // Restituisce l'indirizzo iniziale dello spazio libero,
            // successivo all'intestazione.
            //
            return (void *) (current + (sizeof (mm_head_t)));
          }
        else if (actual_size >= size)
          {
            //
            // Il blocco va usato per intero.
            //
            head->allocated = 1;
            //
            // Restituisce l'indirizzo iniziale dello spazio libero,
            // successivo all'intestazione.
            //
            return (void *) (current + (sizeof (mm_head_t)));
          }
```

```
          }
        //
        // Il blocco è allocato, oppure è di dimensione insufficiente;
        // pertanto occorre passare alla posizione successiva.
        //
        head = (void *) next;
      }
    //
    // Essendo terminato il ciclo precedente, vuol dire
    // che non ci sono spazi disponibili.
    //
    return NULL;
}
```

La funzione ***free()*** libera il blocco di memoria indicato e poi scandisce tutti i blocchi esistenti alla ricerca di quelli liberi che sono adiacenti, per fonderli assieme. Va osservato che la funzione non verifica se il blocco da liberare esiste effettivamente e per evitare errori occorrerebbe una scansione preventiva dei blocchi, a partire dall'inizio.

Listato u9.72. '`./05/lib/free.c`'

```
#include <stdlib.h>
#include <kernel/mm.h>
#include <stdio.h>
void
free (void *ptr)
{
    mm_head_t *start        = (mm_head_t *) os.mem_ph.available_s;
    mm_head_t *head_current = ((mm_head_t *) ptr) - 1;
    mm_head_t *head_next;
    //
    // Verifica il blocco attuale e, se è possibile, lo libera.
    //
    if (head_current->allocated == 1)
      {
        head_current->allocated = 0;
      }
    else
      {
        printf ("[%s] ERROR: cannot free %08X!\n",
                __func__, (uintptr_t) head_current + (sizeof (mm_head_t)));
      }
    //
    // Scandisce i blocchi liberi, cercando quelli adiacenti per
    // allungarli. Se il blocco successivo è il primo, termina,
    // perché non può avvenire alcuna fusione con quello precedente.
    //
```

```
    head_current = start;
    while (true)
      {
        //
        // Individua il blocco successivo.
        //
        head_next = (mm_head_t *) (head_current->next * (sizeof (mm_head_t)));
        //
        // Controlla se è il primo.
        //
        if (head_next == start)
          {
            break;
          }
        //
        //
        //
        if (head_current->allocated == 0)
          {
            //
            // Controlla se si può espandere.
            //
            if (head_next->allocated == 0)
              {
                head_current->next = head_next->next;
              }
            else
              {
                head_current = head_next;
              }
          }
        else
          {
            head_current = (mm_head_t *)
                           (head_current->next * (sizeof (mm_head_t)));
          }
      }
}
```

La funzione ***realloc()***, nel caso sia richiesto un blocco più grande del precedente, si avvale di ***malloc()***, ***memcpy()*** e ***free()***.

Listato u9.73. '`./05/lib/realloc.c`'

```
#include <stdlib.h>
#include <string.h>
#include <kernel/mm.h>
#include <stdio.h>
void
*realloc (void *ptr, size_t size)
{
    uintptr_t  start     = os.mem_ph.available_s;
       size_t  actual_size;
    mm_head_t *head      = ((mm_head_t *) ptr) - 1;
    mm_head_t *head_new;
         void *ptr_new;
    //
    // Verifica che il puntatore riguardi effettivamente
    // un'area occupata.
    //
    if (! head->allocated)
      {
        printf ("[%s] ERROR: cannot re-allocate %08X that is "
                "not already allocated!", __func__, (uintptr_t) ptr);
      }
    //
    // Arrotonda in eccesso il valore di «size», in modo che sia un
    // multiplo della dimensione di «mm_head_t». Altrimenti, la
    // collocazione dei blocchi successivi può avvenire in modo
    // non allineato.
    //
    size = (size + (sizeof (mm_head_t)) -1);
    size = size / (sizeof (mm_head_t));
    size = size * (sizeof (mm_head_t));
    //
    // Determina la dimensione attuale.
    //
    if ((head->next * (sizeof (mm_head_t))) == start)
      {
        actual_size = os.mem_ph.available_e - ((uintptr_t) ptr);
      }
    else
      {
        actual_size = (head->next * (sizeof (mm_head_t))) - ((uintptr_t) ptr);
      }
    //
    // Se la dimensione richiesta è inferiore, può ridurre
    // l'estensione del blocco.
    //
    if (size == actual_size)
      {
```

```
        return ptr;
      }
    else if (size <= (actual_size - (sizeof (mm_head_t)) * 2))
      {
        //
        // Si può ricavare lo spazio libero rimanente.
        //
        head_new = (mm_head_t *) (((char *) ptr) + size);
        //
        head_new->next = head->next;
        head_new->allocated=0;
        //
        head->next = ((uintptr_t) head_new) / (sizeof (mm_head_t));
        //
        return ptr;
      }
    else if (size < actual_size)
      {
        //
        // Anche se è minore, non si può ridurre lo spazio usato
        // effettivamente.
        //
        return ptr;
      }
    else
      {
        //
        // La dimensione richiesta è maggiore.
        //
        ptr_new = malloc (size);
        //
        if (ptr_new)
          {
            //
            // Ricopia i dati nella nuova collocazione.
            //
            memcpy (ptr_new, ptr, actual_size);
            //
            // Libera la collocazione vecchia.
            //
            free (ptr);
            //
            return ptr_new;
          }
        else
          {
            return NULL;
          }
```

```
        }
}
```

Verifica del funzionamento

Per utilizzare le funzioni ***mm_init()*** e ***mm_list()*** occorre aggiornare il file '`kernel_main.c`', aggiungendo in modo particolare delle istruzioni per verificare il funzionamento delle funzioni di allocazione della memoria.

Figura u9.74. Modifiche da apportare al file '`./05/kernel/kernel_main.c`'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
…
        mboot_info (info);
        mboot_show (info);
        //
        // Raccoglie i dati sulla memoria fisica.
        //
        kernel_memory (info);
        //
        // Predispone la tabella GDT.
        //
        gdt ();
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        void *p0 = malloc (0x100);
        void *p1 = malloc (0x1000);
        void *p2 = malloc (0x10000);
                   malloc (0x100000);
        p0 = realloc (p0, 0x1000);
        p1 = realloc (p1, 0x100);
        p2 = realloc (p2, 0x100000);
        free (p1);
        mm_list ();
…
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove si può osservare la mappa della memoria alla fine delle operazioni di allocazione e riallocazione eseguite:

```
05 20070820133728
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010C65C  avail. 0010C65C..01EF0000
[kernel_memory_show] text   00100000..001039D8  rodata 001039D8..00103CD4
[kernel_memory_show] data   00103CD4..00103CD4  bss    00103CE0..0010C65C
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010BD28 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000010000000001000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001000000000
[mm_init] available memory: 31340960 byte
[mm_list] free 0010C660..0010D764 size 00001104
[mm_list] used 0010D768..0011D768 size 00010000
[mm_list] used 0011D76C..0021D76C size 00100000
[mm_list] used 0021D770..0021E770 size 00001000
[mm_list] used 0021E774..0031E774 size 00100000
[mm_list] free 0021E778..01EF0000 size 01BD1888
[kernel_main] system halted
```

Nelle figure successive viene mostrato, schematicamente, ciò che accade. La prima figura mostra lo stato della lista della memoria dopo le prime quattro allocazioni; le figure successive mostrano le riallocazioni che vengono fatte dopo, una a una, per finire con la liberazione della zona associata alla variabile ***p1***. Purtroppo, nelle figure non è stato possibile usare delle proporzioni realistiche.

Figura u9.76. La lista della memoria dopo le prime quattro allocazioni.

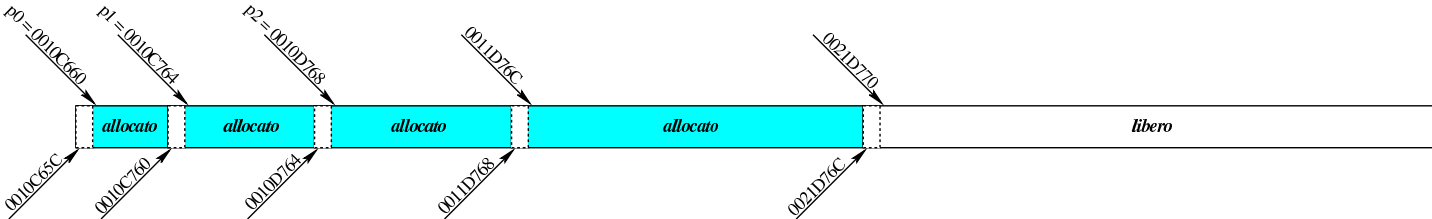



Figura u9.77. La lista della memoria dopo la riallocazione di ***p0***.

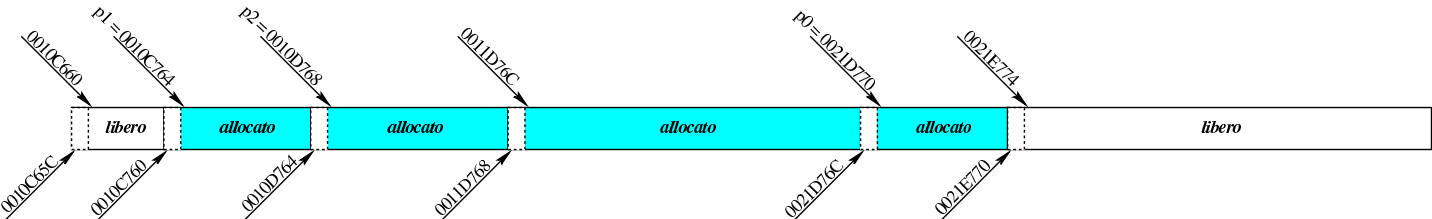



Figura u9.78. La lista della memoria dopo la riallocazione di ***p1***.

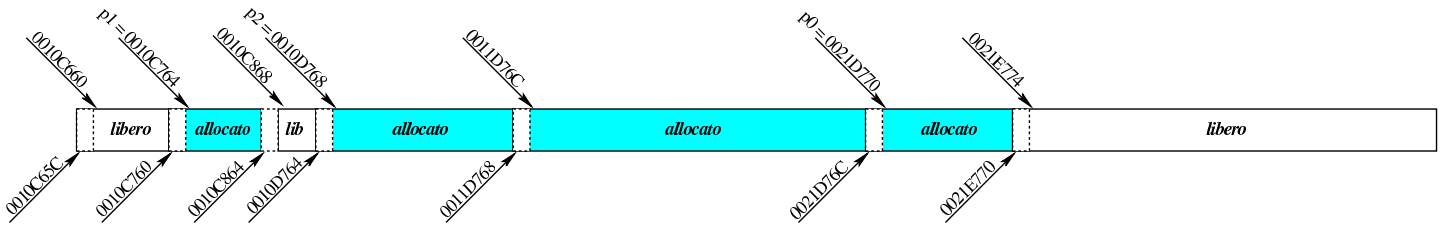



Figura u9.79. La lista della memoria dopo la riallocazione di ***p2***.

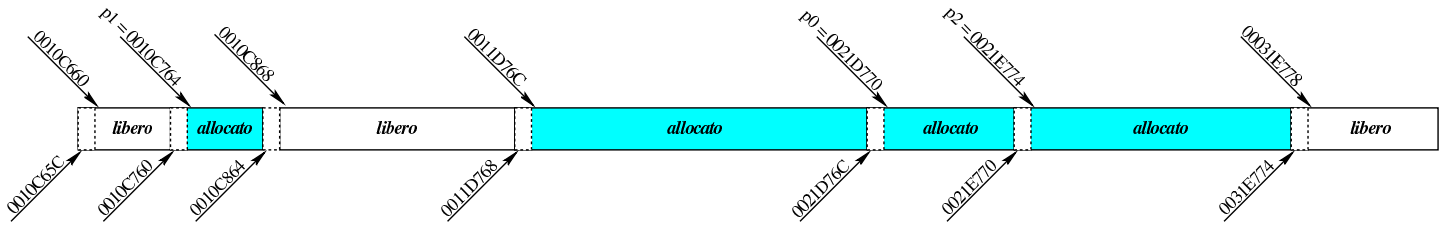



Figura u9.80. La lista della memoria dopo l'eliminazione di ***p1***.

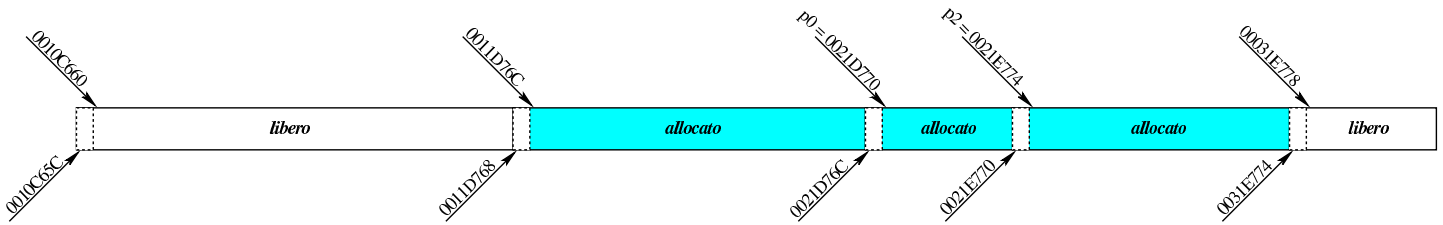



Tabella IDT

In questa fase dello sviluppo del sistema è opportuno predisporre la tabella IDT (*interrupt description table*), con le eccezioni del microprocessore e le interruzioni hardware (IRQ), anche se inizialmente nulla viene gestito effettivamente.

File di intestazione «int.h» e file delle routine di interruzione «isr.s»

Il file di intestazione '`int.h`' contiene la dichiarazione delle funzioni per la gestione delle interruzioni. Viene proposto subito nella sua versione completa, anche se non tutte le funzioni dichiarate vengono presentate immediatamente.

Listato u9.81. '`./05/include/kernel/int.h`'

```
#ifndef _INT_H
#define _INT_H  1

#include <inttypes.h>
#include <stdbool.h>
#include <stdarg.h>
#include <kernel/os.h>

void idt_desc_int     (int       desc,
                       uint32_t offset,
```

```
                        uint16_t selector,
                        bool     present,
                        char     type,
                        char     dpl);

void  idt_load       (void *idtr);
void  idt            (void);
void  irq_remap      (unsigned int offset_1, unsigned int offset_2);
char *exception_name (int exception);
void  idt_print      (void *idtr);

void isr_0  (void);
void isr_1  (void);
void isr_2  (void);
void isr_3  (void);
void isr_4  (void);
void isr_5  (void);
void isr_6  (void);
void isr_7  (void);
void isr_8  (void);
void isr_9  (void);
void isr_10 (void);
void isr_11 (void);
void isr_12 (void);
void isr_13 (void);
void isr_14 (void);
void isr_15 (void);
void isr_16 (void);
void isr_17 (void);
void isr_18 (void);
void isr_19 (void);
void isr_20 (void);
void isr_21 (void);
void isr_22 (void);
void isr_23 (void);
void isr_24 (void);
void isr_25 (void);
void isr_26 (void);
void isr_27 (void);
void isr_28 (void);
void isr_29 (void);
void isr_30 (void);
void isr_31 (void);
void isr_32 (void);
void isr_33 (void);
void isr_34 (void);
void isr_35 (void);
void isr_36 (void);
```

```
void isr_37 (void);
void isr_38 (void);
void isr_39 (void);
void isr_40 (void);
void isr_41 (void);
void isr_42 (void);
void isr_43 (void);
void isr_44 (void);
void isr_45 (void);
void isr_46 (void);
void isr_47 (void);
void isr_128 (void);

void sti (void);
void cli (void);

void isr_exception_unrecoverable (uint32_t eax, uint32_t ecx, uint32_t edx,
                                  uint32_t ebx, uint32_t ebp, uint32_t esi,
                                  uint32_t edi, uint32_t ds,  uint32_t es,
                                  uint32_t fs,  uint32_t gs,
                                  uint32_t interrupt, uint32_t error,
                                  uint32_t eip, uint32_t cs, uint32_t eflags);

void isr_irq (uint32_t eax, uint32_t ecx, uint32_t edx, uint32_t ebx,
              uint32_t ebp, uint32_t esi, uint32_t edi, uint32_t ds,
              uint32_t es,  uint32_t fs,  uint32_t gs,  uint32_t interrupt);

uint32_t isr_syscall (uint32_t start, ...);
uint32_t int_128     (void);

#endif
```

Si può osservare l'elenco delle funzioni '**`isr_`*`n`*`()`**', per la gestione delle varie interruzioni catalogate nella tabella IDT. In particolare, l'interruzione 128_{10}, ovvero 80_{16}, viene usata per le chiamate di sistema. Queste funzioni sono dichiarate formalmente nel file '`isr.s`' che viene mostrato integralmente nel listato successivo.

Listato u9.82. '`./05/lib/int/isr.s`'

```
.extern isr_exception_unrecoverable
.extern isr_irq
.extern isr_syscall
#
.globl isr_0
.globl isr_1
.globl isr_2
.globl isr_3
.globl isr_4
```

```
.globl isr_5
.globl isr_6
.globl isr_7
.globl isr_8
.globl isr_9
.globl isr_10
.globl isr_11
.globl isr_12
.globl isr_13
.globl isr_14
.globl isr_15
.globl isr_16
.globl isr_17
.globl isr_18
.globl isr_19
.globl isr_20
.globl isr_21
.globl isr_22
.globl isr_23
.globl isr_24
.globl isr_25
.globl isr_26
.globl isr_27
.globl isr_28
.globl isr_29
.globl isr_30
.globl isr_31
.globl isr_32
.globl isr_33
.globl isr_34
.globl isr_35
.globl isr_36
.globl isr_37
.globl isr_38
.globl isr_39
.globl isr_40
.globl isr_41
.globl isr_42
.globl isr_43
.globl isr_44
.globl isr_45
.globl isr_46
.globl isr_47
.globl isr_128

    ########################################################################
    # Nella pila è già stato inserito dal microprocessore:                 #
    #   [omissis]                                                          #
    #   push %eflags                                                       #
```

```
    #   push %cs                                                            #
    #   push %eip                                                           #
    #########################################################################

isr_0:          # «division by zero exception»
    cli
    push $0     # Codice di errore fittizio.
    push $0     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_1:          # «debug exception»
    cli
    push $0     # Codice di errore fittizio.
    push $1     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_2:          # «non maskable interrupt exception»
    cli
    push $0     # Codice di errore fittizio.
    push $2     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_3:          # «breakpoint exception»
    cli
    push $0     # Codice di errore fittizio.
    push $3     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_4:          # «into detected overflow exception»
    cli
    push $0     # Codice di errore fittizio.
    push $4     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_5:          # «out of bounds exception»
    cli
    push $0     # Codice di errore fittizio.
    push $5     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_6:          # «invalid opcode exception»
    cli
    push $0     # Codice di errore fittizio.
    push $6     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_7:          # «no coprocessor exception»
    cli
    push $0     # Codice di errore fittizio.
```

```
    push $7     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_8:          # «double fault exception»
    cli
    #
    push $8     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_9:          # «coprocessor segment overrun exception»
    cli
    push $0     # Codice di errore fittizio.
    push $9     # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_10:         # «bad TSS exception»
    cli
    #
    push $10    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_11:         # «segment not present exception»
    cli
    #
    push $11    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_12:         # «stack fault exception»
    cli
    #
    push $12    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_13:         # «general protection fault exception»
    cli
    #
    push $13    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_14:         # «page fault exception»
    cli
    #
    push $14    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_15:         # «unknown interrupt exception»
    cli
    push $0     # Codice di errore fittizio.
    push $15    # Numero dell'eccezione.
```

```
    jmp exception_unrecoverable
#
isr_16:         # «coprocessor fault exception»
    cli
    push $0     # Codice di errore fittizio.
    push $16    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_17:         # «alignment check exception»
    cli
    push $0     # Codice di errore fittizio.
    push $17    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_18:         # «machine check exception»
    cli
    push $0     # Codice di errore fittizio.
    push $18    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_19:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $19    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_20:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $20    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_21:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $21    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_22:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $22    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_23:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $23    # Numero dell'eccezione.
    jmp exception_unrecoverable
```

```
#
isr_24:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $24    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_25:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $25    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_26:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $26    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_27:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $27    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_28:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $28    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_29:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $29    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_30:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $30    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
isr_31:         # «reserved exception»
    cli
    push $0     # Codice di errore fittizio.
    push $31    # Numero dell'eccezione.
    jmp exception_unrecoverable
#
```

```
isr_32:         # IRQ 0: «timer»
    cli
    push $0     # Codice di errore fittizio.
    push $32    # Numero IRQ + 32.
    jmp irq
#
isr_33:         # IRQ 1: tastiera
    cli
    push $0     # Codice di errore fittizio.
    push $33    # Numero IRQ + 32.
    jmp irq
#
isr_34:         # IRQ 2: viene attivato per gli IRQ da 8 a 15.
    cli
    push $0     # Codice di errore fittizio.
    push $34    # Numero IRQ + 32.
    jmp irq
#
isr_35:         # IRQ 3
    cli
    push $0     # Codice di errore fittizio.
    push $35    # Numero IRQ + 32.
    jmp irq
#
isr_36:         # IRQ 4
    cli
    push $0     # Codice di errore fittizio.
    push $36    # Numero IRQ + 32.
    jmp irq
#
isr_37:         # IRQ 5
    cli
    push $0     # Codice di errore fittizio.
    push $37    # Numero IRQ + 32.
    jmp irq
#
isr_38:         # IRQ 6: unità a dischetti
    cli
    push $0     # Codice di errore fittizio.
    push $38    # Numero IRQ + 32.
    jmp irq
#
isr_39:         # IRQ 7: LPT 1
    cli
    push $0     # Codice di errore fittizio.
    push $39    # Numero IRQ + 32.
    jmp irq
#
isr_40:         # IRQ 8: «real time clock (RTC)»
```

```
    cli
    push $0      # Codice di errore fittizio.
    push $40     # Numero IRQ + 32.
    jmp irq
#
isr_41:          # IRQ 9
    cli
    push $0      # Codice di errore fittizio.
    push $41     # Numero IRQ + 32.
    jmp irq
#
isr_42:          # IRQ 10
    cli
    push $0      # Codice di errore fittizio.
    push $42     # Numero IRQ + 32.
    jmp irq
#
isr_43:          # IRQ 11
    cli
    push $0      # Codice di errore fittizio.
    push $43     # Numero IRQ + 32.
    jmp irq
#
isr_44:          # IRQ 12: mouse PS/2
    cli
    push $0      # Codice di errore fittizio.
    push $44     # Numero IRQ + 32.
    jmp irq
#
isr_45:          # IRQ 13: coprocessore matematico
    cli
    push $0      # Codice di errore fittizio.
    push $45     # Numero IRQ + 32.
    jmp irq
#
isr_46:          # IRQ 14: canale IDE primario
    cli
    push $0      # Codice di errore fittizio.
    push $46     # Numero IRQ + 32.
    jmp irq
#
isr_47:          # IRQ 15: canale IDE secondario
    cli
    push $0      # Codice di errore fittizio.
    push $47     # Numero IRQ + 32.
    jmp irq
#
isr_128:         # Chiamate di sistema.
    cli
```

```
    call isr_syscall
    iret
#
# Eccezioni che per il momento non sono gestibili.
#
exception_unrecoverable:

    ########################################################################
    # A questo punto, nella pila sono stati aggiunti:                      #
    #   push $<n_errore>                                                   #
    #   push $<n_voce_idt>                                                 #
    ########################################################################

    pushl %gs
    pushl %fs
    pushl %es
    pushl %ds
    pushl %edi
    pushl %esi
    pushl %ebp
    pushl %ebx
    pushl %edx
    pushl %ecx
    pushl %eax
    #
    call isr_exception_unrecoverable
    #
    popl %eax
    popl %ecx
    popl %edx
    popl %ebx
    popl %ebp
    popl %esi
    popl %edi
    popl %ds
    popl %es
    popl %fs
    popl %gs
    add $4, %esp        # espelle il numero dell'eccezione
    add $4, %esp        # espelle il codice di errore
    #
    iret
#
# IRQ hardware.
#
irq:

    ########################################################################
    # A questo punto, nella pila sono stati aggiunti:                      #
```

```
    #   push $0                                                          #
    #   push $<n_voce_idt>                                               #
    ####################################################################

    pushl %gs
    pushl %fs
    pushl %es
    pushl %ds
    pushl %edi
    pushl %esi
    pushl %ebp
    pushl %ebx
    pushl %edx
    pushl %ecx
    pushl %eax
    #
    call isr_irq
    #
    popl %eax
    popl %ecx
    popl %edx
    popl %ebx
    popl %ebp
    popl %esi
    popl %edi
    popl %ds
    popl %es
    popl %fs
    popl %gs
    add $4, %esp        # espelle il numero dell'interruzione
    add $4, %esp        # espelle il codice di errore fittizio.
    #
    iret
#
```

Funzioni per definire la tabella IDT

«

Per facilitare la compilazione della tabella IDT viene usata la funzione ***idt_desc_int()***, con la quale si deve specificare il numero del descrittore della tabella e i dati da inserirvi. La tabella IDT è definita nella variabile strutturata ***os.idt***, dichiarata nel file '`os.h`'.

Listato u9.83. '`./05/lib/int/idt_desc_int.c`'

```
#include <kernel/int.h>
void
idt_desc_int (int       desc,
              uint32_t offset,
              uint16_t selector,
              bool     present,
              char     type,
              char     dpl)
{
    //
    // Azzera i bit riservati e quello di sistema.
    //
    os.idt[desc].filler = 0;
    os.idt[desc].system = 0;
    //
    // Indirizzo relativo.
    //
    os.idt[desc].offset_a = (offset & 0x0000FFFF);
    os.idt[desc].offset_b = (offset / 0x10000);
    //
    // Selettore.
    //
    os.idt[desc].selector = selector;
    //
    // Voce valida o meno.
    //
    os.idt[desc].present = present;
    //
    // Tipo (gate type).
    //
    os.idt[desc].type = (type & 0x0F);
    //
    // DPL.
    //
    os.idt[desc].dpl = (dpl & 0x03);
}
```

Per verificare il contenuto della tabella IDT viene predisposta la funzione ***idt_print()*** che richiede come parametro il puntatore all'area di memoria che descrive il registro ***IDTR***. Così come viene proposta, la funzione mostra il contenuto completo della tabella IDT, ma questo supera generalmente le righe visualizzabili sullo schermo; pertanto, in caso di necessità, la funzione va modificata in modo da mostrare solo la porzione di proprio interesse.

Listato u9.84. '`./05/lib/int/idt_print.c`'

```
#include <kernel/int.h>
#include <stdio.h>
//
// Mostra il contenuto di una tabella IDT, a partire dal puntatore al
// registro IDTR in memoria. Pertanto non si avvale, volutamente, della
// struttura già predisposta con il linguaggio C, mentre «local_idtr_t»
// viene creata qui solo provvisoriamente, per uso interno. Ciò serve ad
// assicurare che questa funzione compia il proprio lavoro in modo
// indipendente, garantendo la visualizzazione di dati reali.
//
typedef struct {
    uint16_t  limit;
    uint32_t  base;
} __attribute__ ((packed)) local_idtr_t;
//
void
idt_print (void *idtr)
{
    local_idtr_t *g = idtr;
        uint32_t *p = (uint32_t *) g->base;

    int max = (g->limit + 1) / (sizeof (uint32_t));
    int i;

    for (i = 0; i < max; i+=2)
      {
        printf ("[%s] %02" PRIx32 " %08X %08X\n",
                __func__, i/2, p[i], p[i+1]);
      }
}
```

La funzione ***irq_remap()*** è necessaria per rimappare le interruzioni hardware nella tabella IDT, in modo che non intralcino quelle associate alle eccezioni. La funzione richiede l'indicazione del numero iniziale di interruzione per i due gruppi di IRQ (da IRQ 0 a IRQ 7 e da IRQ 8 a IRQ 15). Successivamente, nella funzione ***idt()***, viene

usata ***irq_remap()*** in modo da rimappare le interruzioni hardware a partire da 32, per finire a 47.

Listato u9.85. '`./05/lib/int/irq_remap.c`'

```
#include <kernel/int.h>
#include <stdio.h>
void
irq_remap (unsigned int offset_1, unsigned int offset_2)
{
    //
    // PIC_P è il PIC primario o «master»;
    // PIC_S è il PIC secondario o «slave».
    //
    // Quando si manifesta un IRQ che riguarda il PIC secondario,
    // il PIC primario riceve IRQ 2.
    //
    // ICW = initialization command word.
    // OCW = operation command word.
    //
    printf ("[%s] PIC (programmable interrupt controller) remap: ", __func__);

    outb (0x20, 0x10 + 0x01);   // Inizializzazione: 0x10 significa che
    outb (0xA0, 0x10 + 0x01);   // si tratta di ICW1; 0x01 significa che
    printf ("ICW1");            // si deve arrivare fino a ICW4.

    outb (0x21, offset_1);      // ICW2: PIC_P a partire da «offset_1».
    outb (0xA1, offset_2);      //       PIC_S a partire da «offset_2».
    printf (", ICW2");
    outb (0x21, 0x04);          // ICW3 PIC_P: IRQ2 pilotato da PIC_S.
    outb (0xA1, 0x02);          // ICW3 PIC_S: pilota IRQ2 di PIC_P.
    printf (", ICW3");
    outb (0x21, 0x01);          // ICW4: si precisa solo la modalità
    outb (0xA1, 0x01);          // del microprocessore; 0x01 = 8086.
    printf (", ICW4");

    outb (0x21, 0x00);          // OCW1: azzera la maschera in modo da
    outb (0xA1, 0x00);          // abilitare tutti i numeri IRQ.
    printf (", OCW1.\n");
}
```

Per caricare la tabella IDT dichiarata in memoria, occorre predisporre la copia del registro ***IDTR*** con i riferimenti necessari a raggiungerla, quindi va usata l'istruzione '**`LIDT`**', con il linguaggio assemblatore. La funzione ***idt_load()*** viene usata per pilotare l'istruzione '**`LIDT`**'.

Listato u9.86. '`./05/lib/int/idt_load.s`'

```
.globl  idt_load
#
idt_load:
    enter $0, $0
    .equ idtr_pointer,  8          # Primo argomento.
    mov  idtr_pointer(%ebp), %eax  # Copia il puntatore
                                   # in EAX.
    leave
    #
    lidt (%eax)   # Utilizza la tabella IDT a cui punta EAX.
    #
    ret
```

La funzione ***idt()*** utilizza le altre descritte in questa sezione, per mettere in funzione la gestione delle interruzioni.

Listato u9.87. '`./05/lib/int/idt.c`'

```
#include <kernel/int.h>
void
idt (void)
{
    //
    // Imposta i dati necessari al registro IDTR.
    //
    os.idtr.limit = (sizeof (os.idt) - 1);
    os.idtr.base  = (uint32_t) &os.idt[0];
    //
    // Azzera le voci previste dell'array «os.idt[]».
    //
    int i;
    for (i = 0; i < ((sizeof (os.idt)) / 8); i++)
      {
        idt_desc_int (i, 0, 0, 0, 0, 0);
      }
    //
    // Associa le interruzioni hardware da IRQ 0 a IRQ 7
    // a partire dal descrittore 32 e quelle da IRQ 8 a
    // IRQ 15, a partire dal descrittore 40.
```

```
//
irq_remap (32, 40);
//
// Associa le routine ISR ai descrittori della tabella
// IDT.
//
idt_desc_int ( 0, (uint32_t) isr_0,  0x0008, 1, 0xE, 0);
idt_desc_int ( 1, (uint32_t) isr_1,  0x0008, 1, 0xE, 0);
idt_desc_int ( 2, (uint32_t) isr_2,  0x0008, 1, 0xE, 0);
idt_desc_int ( 3, (uint32_t) isr_3,  0x0008, 1, 0xE, 0);
idt_desc_int ( 4, (uint32_t) isr_4,  0x0008, 1, 0xE, 0);
idt_desc_int ( 5, (uint32_t) isr_5,  0x0008, 1, 0xE, 0);
idt_desc_int ( 6, (uint32_t) isr_6,  0x0008, 1, 0xE, 0);
idt_desc_int ( 7, (uint32_t) isr_7,  0x0008, 1, 0xE, 0);
idt_desc_int ( 8, (uint32_t) isr_8,  0x0008, 1, 0xE, 0);
idt_desc_int ( 9, (uint32_t) isr_9,  0x0008, 1, 0xE, 0);
idt_desc_int (10, (uint32_t) isr_10, 0x0008, 1, 0xE, 0);
idt_desc_int (11, (uint32_t) isr_11, 0x0008, 1, 0xE, 0);
idt_desc_int (12, (uint32_t) isr_12, 0x0008, 1, 0xE, 0);
idt_desc_int (13, (uint32_t) isr_13, 0x0008, 1, 0xE, 0);
idt_desc_int (14, (uint32_t) isr_14, 0x0008, 1, 0xE, 0);
idt_desc_int (15, (uint32_t) isr_15, 0x0008, 1, 0xE, 0);
idt_desc_int (16, (uint32_t) isr_16, 0x0008, 1, 0xE, 0);
idt_desc_int (17, (uint32_t) isr_17, 0x0008, 1, 0xE, 0);
idt_desc_int (18, (uint32_t) isr_18, 0x0008, 1, 0xE, 0);
idt_desc_int (19, (uint32_t) isr_19, 0x0008, 1, 0xE, 0);
idt_desc_int (20, (uint32_t) isr_20, 0x0008, 1, 0xE, 0);
idt_desc_int (21, (uint32_t) isr_21, 0x0008, 1, 0xE, 0);
idt_desc_int (22, (uint32_t) isr_22, 0x0008, 1, 0xE, 0);
idt_desc_int (23, (uint32_t) isr_23, 0x0008, 1, 0xE, 0);
idt_desc_int (24, (uint32_t) isr_24, 0x0008, 1, 0xE, 0);
idt_desc_int (25, (uint32_t) isr_25, 0x0008, 1, 0xE, 0);
idt_desc_int (26, (uint32_t) isr_26, 0x0008, 1, 0xE, 0);
idt_desc_int (27, (uint32_t) isr_27, 0x0008, 1, 0xE, 0);
idt_desc_int (28, (uint32_t) isr_28, 0x0008, 1, 0xE, 0);
idt_desc_int (29, (uint32_t) isr_29, 0x0008, 1, 0xE, 0);
idt_desc_int (30, (uint32_t) isr_10, 0x0008, 1, 0xE, 0);
idt_desc_int (31, (uint32_t) isr_31, 0x0008, 1, 0xE, 0);
```

```
    idt_desc_int (32, (uint32_t) isr_32, 0x0008, 1, 0xE, 0);
    idt_desc_int (33, (uint32_t) isr_33, 0x0008, 1, 0xE, 0);
    idt_desc_int (34, (uint32_t) isr_34, 0x0008, 1, 0xE, 0);
    idt_desc_int (35, (uint32_t) isr_35, 0x0008, 1, 0xE, 0);
    idt_desc_int (36, (uint32_t) isr_36, 0x0008, 1, 0xE, 0);
    idt_desc_int (37, (uint32_t) isr_37, 0x0008, 1, 0xE, 0);
    idt_desc_int (38, (uint32_t) isr_38, 0x0008, 1, 0xE, 0);
    idt_desc_int (39, (uint32_t) isr_39, 0x0008, 1, 0xE, 0);
    idt_desc_int (40, (uint32_t) isr_40, 0x0008, 1, 0xE, 0);
    idt_desc_int (41, (uint32_t) isr_41, 0x0008, 1, 0xE, 0);
    idt_desc_int (42, (uint32_t) isr_42, 0x0008, 1, 0xE, 0);
    idt_desc_int (43, (uint32_t) isr_43, 0x0008, 1, 0xE, 0);
    idt_desc_int (44, (uint32_t) isr_44, 0x0008, 1, 0xE, 0);
    idt_desc_int (45, (uint32_t) isr_45, 0x0008, 1, 0xE, 0);
    idt_desc_int (46, (uint32_t) isr_46, 0x0008, 1, 0xE, 0);
    idt_desc_int (47, (uint32_t) isr_47, 0x0008, 1, 0xE, 0);
    //
    // Questo è per le chiamate di sistema.
    //
    idt_desc_int (128, (uint32_t) isr_128, 0x0008, 1, 0xE,
                  0);
    //
    // Rende operativa la tabella con le eccezioni e gli
    // IRQ.
    //
    idt_load (&os.idtr);
    //
    // Abilita le interruzioni hardware (IRQ).
    //
    sti ();
}
```

Gestione delle interruzioni

Le funzioni ‘**`isr_`*n*`()`**’ si limitano a chiamare altre funzioni scritte in linguaggio C, per la gestione delle eccezioni, delle interruzioni hardware e per le chiamate di sistema. In questa fase vengono mostrate le funzioni per la gestione delle eccezioni, anche se in forma estremamente limitata, e si propongono temporaneamente delle funzioni fittizie per la gestione degli altri casi.

Listato u9.88. './05/lib/int/isr_exception_unrecoverable.c'

```
#include <kernel/int.h>
#include <stdio.h>
void
isr_exception_unrecoverable (uint32_t eax, uint32_t ecx,
                             uint32_t edx, uint32_t ebx,
                             uint32_t ebp, uint32_t esi,
                             uint32_t edi, uint32_t ds,
                             uint32_t es, uint32_t fs,
                             uint32_t gs,
                             uint32_t interrupt,
                             uint32_t error, uint32_t eip,
                             uint32_t cs,
                             uint32_t eflags)
{
    printf ("[%s] ERROR: exception %i: \"%s\"\n",
            __func__, interrupt,
            exception_name (interrupt));
    //
    _Exit (0);
}
```

La funzione ***isr_exception_unrecoverable()***, appena mostrata, viene chiamata dal file 'isr.s', per le interruzioni che riguardano le eccezioni. La funzione si limita a visualizzare un messaggio di errore e a fermare il sistema. Per visualizzare il tipo di eccezione che si è verificato si avvale della funzione ***exception_name()*** che appare nel listato successivo.

Listato u9.89. './05/lib/int/exception_name.c'

```
#include <kernel/int.h>
char
*exception_name (int exception)
{
    char *description[19] = {"division by zero",
                             "debug",
                             "non maskable interrupt",
                             "breakpoint",
                             "into detected overflow",
                             "out of bounds",
```

```
                                  "invalid opcode",
                                  "no coprocessor",
                                  "double fault",
                                  "coprocessor segmento overrun",
                                  "bad TSS",
                                  "segment not present",
                                  "stack fault",
                                  "general protection fault",
                                  "page fault",
                                  "unknown interrupt",
                                  "coprocessor fault",
                                  "alignment check",
                                  "machine check"};
    //
    if (exception >= 0 && exception <= 18)
      {
        return description[exception];
      }
    else
      {
        return "unknown";
      }
}
```

A proposito della funzione ***exception_name()*** va osservata la particolarità del comportamento del compilatore GNU C, il quale utilizza, senza che ciò sia stato richiesto espressamente, la funzione standard ***memcpy()***. Pertanto, tale funzione deve essere disponibile, altrimenti, in fase di collegamento (*link*) la compilazione fallisce.

Per la gestione delle interruzioni hardware è competente la funzione ***isr_irq()***, ma per il momento viene proposta una versione provvisoria, priva di alcuna gestione, dove ci si limita a inviare il messaggio «EOI» ai PIC (*programmable interrupt controller*) coinvolti.

Listato u9.90. Una prima versione del file '`./05/lib/int/isr_irq.c`'

```
#include <kernel/int.h>
#include <kernel/io.h>
void
isr_irq (uint32_t eax, uint32_t ecx, uint32_t edx,
         uint32_t ebx, uint32_t ebp, uint32_t esi,
```

```
         uint32_t edi, uint32_t ds, uint32_t es,
         uint32_t fs,  uint32_t gs,  uint32_t interrupt)
{
    int irq = interrupt - 32;
    //
    // Finito il compito della funzione che deve reagire
    // all'interruzione IRQ, occorre informare i PIC
    // (programmable interrupt controller).
    //
    // Se il numero IRQ è tra 8 e 15, manda un messaggio
    // «EOI»
    // (End of IRQ) al PIC 2.
    //
    if (irq >= 8)
      {
        outb (0xA0, 0x20);
      }
    //
    // Poi manda un messaggio «EOI» al PIC 1.
    //
    outb (0x20, 0x20);
}
```

Anche la funzione ***isr_syscall()*** che dovrebbe prendersi cura delle chiamate di sistema, viene proposta inizialmente priva di alcun effetto.

Listato u9.91. Una prima versione del file '`./05/lib/int/isr_syscall.c`'

```
#include <kernel/int.h>
uint32_t
isr_syscall (uint32_t start, ...)
{
    return 0;
}
```

Piccole funzioni di contorno

Per facilitare l'accesso alle istruzioni '**STI**' e '**CLI**' del linguaggio assemblatore, vengono predisposte due funzioni con lo stesso nome.

Listato u9.92. './05/lib/int/cli.s'

```
.globl cli
#
cli:
    cli
    ret
```

Listato u9.93. './05/lib/int/sti.s'

```
.globl sti
#
sti:
    sti
    ret
```

Verifica del funzionamento

Per verificare il lavoro svolto fino a questo punto, è necessario sviluppare ulteriormente i file 'kernel_main.c', dove in particolare si va a produrre un errore che causa un eccezione dovuta a una divisione per zero.

Figura u9.94. Modifiche da apportare al file './05/kernel/kernel_main.c'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
…
        kernel_memory (info);
        //
        // Predispone la tabella GDT.
        //
        gdt ();
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        //
        // Omissis.
```

```
        //
        //
        // Predispone la tabella IDT.
        //
        idt();
        //
        // Crea un errore volontario.
        //
        int x = 3;
        x = 7 / (x - 3);        // x = 7 / 0
...
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove alla fine si vede la segnalazione di errore dovuta alla divisione per zero:

```
05 20070821144531
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010D8BC  avail. 0010D8BC..01EF0000
[kernel_memory_show] text   00100000..001049DC  rodata 001049E0..00104F34
[kernel_memory_show] data   00104F34..00104F34  bss    00104F40..0010D8BC
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010CF88 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000010000000001000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001000000000
[mm_init] available memory: 31336256 byte
[irq_remap] PIC (programmable interrupt controller) remap: ICW1, ICW2, ICW3,
ICW4, OCW1.
[isr_exception_unrecoverable] ERROR: exception 0: "division by zero"
```

Chiamate di sistema

«

Nel sistema in corso di realizzazione sono previste le chiamate di sistema, anche se in pratica sono inutili, dal momento che non è possibile gestire processi elaborativi indipendenti. Queste chiamate si ottengono mettendo gli argomenti nella pila e utilizzando l'interruzione 128 (ovvero 80_{16}). Si osservi che questo meccanismo è diverso da quello usato dal kernel Linux, dove gli argomenti sono passati normalmente attraverso i registri del microprocessore.

Il punto di inizio per una chiamata di sistema è la funzione ***syscall()***, con la quale va indicato il numero della chiamata, seguito dagli argomenti necessari, in base al contesto.

Listato u9.96. './05/lib/sys/syscall.c'

```
#include <sys/syscall.h>
#include <kernel/int.h>
uint32_t
syscall (int n, ...)
{
    return int_128 ();
}
```

Come si vede, ci si limita a utilizzare la funzione ***int_128()***, scritta però in linguaggio assemblatore, come si vede nel listato successivo.

Listato u9.97. './05/lib/int/int_128.s'

```
.globl int_128
#
int_128:
    int $128
    ret
```

Questa doppia mediazione ha delle conseguenze nella composizione della pila dei dati, al momento dell'avvio della funzione che deve trattare l'interruzione.

File di intestazione «syscall.h»

«

Il file di intestazione 'syscall.h' dichiara le funzioni usate per generare una chiamata di sistema e poi per eseguirla; inoltre, si definiscono delle macro-variabili per dare un nome alle chiamate che in realtà sono indicate solo per numero.

Listato u9.98. './05/include/sys/syscall.h'

```
#ifndef _SYSCALL_H
#define _SYSCALL_H      1

#include <inttypes.h>
#include <stdarg.h>

#define SYSCALL_malloc       1
#define SYSCALL_realloc      2
#define SYSCALL_free         3
#define SYSCALL_console_putc 4

uint32_t syscall  (int n, ...);
uint32_t vsyscall (int n, va_list ap);

#endif
```

«

Una volta provocata l'interruzione 128, si ottiene l'attivazione della funzione ***isr_128()***, la quale avvia a sua volta la funzione ***isr_syscall()*** che deve provvedere a ripescare gli argomenti della chiamata originale, quindi avvia la funzione che può elaborarli: ***vsyscall()***.

Listato u9.99. '`./05/lib/int/isr_syscall.c`'

```
#include <kernel/int.h>
#include <sys/syscall.h>
uint32_t
isr_syscall (uint32_t start, ...)
{
    va_list ap;
    uint32_t value;
    //
    // Colloca il puntatore all'inizio.
    //
    va_start (ap, start);
    //
    // Salta i dati che non servono.
    //
    value = va_arg (ap, uint32_t); // CS
    value = va_arg (ap, uint32_t); // EFLAGS
    value = va_arg (ap, uint32_t); // ???
    value = va_arg (ap, uint32_t); // ESP
    value = va_arg (ap, uint32_t); // SS
    value = va_arg (ap, uint32_t); // EIP
    value = va_arg (ap, uint32_t); // EIP
    value = va_arg (ap, uint32_t); // n. chiamata
    //
    // Attualmente «ap» punta all'argomento successivo
    // al numero di chiamata.
    //
    return vsyscall (value, ap);
}
```

Listato u9.100. '`./05/lib/sys/vsyscall.c`'

```
#include <sys/syscall.h>
#include <stdint.h>
#include <inttypes.h>
#include <stdlib.h>
#include <stdarg.h>
#include <kernel/vga.h>

uint32_t
vsyscall (int n, va_list ap)
{
    if (n == SYSCALL_malloc)
      {
        size_t  size = va_arg (ap, size_t);
        return (uint32_t) malloc (size);
      }
    else if (n == SYSCALL_realloc)
      {
        void   *ptr  = va_arg (ap, void*);   // Qui, «void*» va scritto
        size_t  size = va_arg (ap, size_t);  // attaccato e senza parentesi.
        return (uint32_t) realloc (ptr, size);
      }
    else if (n == SYSCALL_free)
      {
        void   *ptr  = va_arg (ap, void*);
        free (ptr);
        return 0;
      }
    else if (n == SYSCALL_console_putc)
      {
        int     c    = va_arg (ap, int);
        vga_putc (c);
        return (uint32_t) c;
      }
    else
      {
        printf ("[%s] ERROR: unknown syscall: %i!\n", __func__, n);
        return -1;
      }
}
```

Verifica del funzionamento

Per verificare il funzionamento delle chiamate di sistema, si può modificare il file '`kernel_main.c`' nel modo seguente, allo scopo di visualizzare sullo schermo la parola «Ciao».

Figura u9.101. Modifiche da apportare al file '`./05/kernel/kernel_main.c`'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
#include <sys/syscall.h>
...
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        //
        // Omissis.
        //
        //
        // Predispone la tabella IDT.
        //
        idt();
        //
        // Prova le chiamate di sistema.
        //
        syscall (SYSCALL_console_putc, 'C');
        syscall (SYSCALL_console_putc, 'i');
        syscall (SYSCALL_console_putc, 'a');
        syscall (SYSCALL_console_putc, 'o');
        syscall (SYSCALL_console_putc, '\n');
...
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove prima della conclusione si vede l'emissione della parola «Ciao»:

```
05 20070821155848
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010DADC  avail. 0010DADC..01EF0000
[kernel_memory_show] text   00100000..00104BA8  rodata 00104BC0..00105144
[kernel_memory_show] data   00105144..00105144  bss    00105160..0010DADC
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010D1A8 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000010000000001000000000000
[gdt_print] 1 00000000000000000011110111110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000011110111110000 00000000110000001001001000000000
[mm_init] available memory: 31335712 byte
[irq_remap] PIC (programmable interrupt controller) remap: ICW1, ICW2, ICW3,
ICW4, OCW1.
Ciao
[kernel_main] system halted
```

Interruzioni hardware

Le interruzioni hardware che vengono gestite in questo sistema sono solo IRQ 0 (temporizzatore o *timer*) e IRQ 1 (tastiera). Il file '`isr_irq.c`' che in precedenza è stato ridotto per sospendere il problema delle interruzioni hardware ha la forma finale del listato successivo.

Listato u9.103. '`./05/lib/int/isr_irq.c`'

```
#include <kernel/int.h>
#include <kernel/io.h>
#include <kernel/timer.h>
#include <kernel/keyboard.h>
void
isr_irq (uint32_t eax, uint32_t ecx, uint32_t edx, uint32_t ebx,
         uint32_t ebp, uint32_t esi, uint32_t edi, uint32_t ds,
         uint32_t es,  uint32_t fs,  uint32_t gs,  uint32_t interrupt)
{
    int irq = interrupt - 32;
    //
    //
    //
    switch (irq)
      {
        case  0: timer (); break;
        case  1: keyboard (); break;
      }
    //
    // Finito il compito della funzione che deve reagire all'interruzione
    // IRQ, occorre informare i PIC (programmable interrupt controller).
```

```
    //
    // Se il numero IRQ è tra 8 e 15, manda un messaggio «EOI»
    // (End of IRQ) al PIC 2.
    //
    if (irq >= 8)
      {
        outb (0xA0, 0x20);
      }
    //
    // Poi manda un messaggio «EOI» al PIC 1.
    //
    outb (0x20, 0x20);
}
```

Gestione del temporizzatore

La gestione del temporizzatore è raccolta dalla libreria che fa capo al file di intestazione '`timer.h`' come appare nel listato successivo.

Listato u9.104. '`./05/include/kernel/timer.h`'

```
#ifndef _TIMER_H
#define _TIMER_H        1

#include <time.h>
#include <kernel/os.h>

void timer      (void);
void timer_freq (clock_t freq);

#endif
```

Il temporizzatore genera impulsi a una frequenza data e a ogni impulso produce un'interruzione. Per regolare tale frequenza occorre comunicare con le porte 43_{16} e 40_{16}, inviando il divisore da applicare alla frequenza di riferimento che è 1,193 181 MHz. La funzione ***timer_freq()*** stabilisce la frequenza da generare, calcolando il divisore da applicare.[1]

Listato u9.105. '`./05/lib/timer/timer_freq.c`'

```
#include <kernel/timer.h>
#include <stdint.h>
#include <stdio.h>
void
timer_freq (clock_t freq)
{
    int input_freq = 1193180;
    //
    // La frequenza di riferimento è 1,19318 MHz, la quale va
    // divisa per la frequenza che si intende avere effettivamente.
    //
    int divisor = input_freq / freq;
    //
    // Il risultato deve essere un valore intero maggiore di zero
    // e inferiore di UINT16_MAX, altrimenti è stata chiesta una
    // frequenza troppo elevata o troppo bassa.
    //
    if (divisor == 0 || divisor > UINT16_MAX)
      {
        printf ("[%s] ERROR: IRQ 0 frequency wrong: %i Hz!\n"
                "[%s]        The min allowed frequency is 18.22 Hz.\n",
                "[%s]        The max allowed frequency is 1.19 MHz.\n",
                __func__, freq, __func__, __func__);
        return;
      }
    //
    // Il valore che si ottiene, ovvero il «divisore», va
    // comunicato al PIT (programmable interval timer),
    // spezzandolo in due parti.
    //
    outb (0x43, 0x36);
    outb (0x40, divisor & 0x0F);        // Byte inferiore del numero.
    outb (0x40, divisor / 0x10);        // Byte superiore del numero.
    //
    // Annota la frequenza attuale degli impulsi provenienti dal
    // PIT (programmable interval timer).
    //
    os.timer.freq = freq;
}
```

La funzione ***timer()*** è quella che viene eseguita automaticamente, ogni volta che si presenta un'interruzione che deriva da un IRQ 0. Di norma lo scopo di una funzione di questo tipo è controllare la gestione corretta dei processi, ma in mancanza di questi, si potrebbero avviare delle funzioni che assicurano un'esecuzione brevissima, salvo il verificarsi di eventi specifici. Nel listato successivo si presenta una funzione ***timer()*** praticamente vuota e i file di intestazione incorporati sono ipotetici.

Listato u9.106. './05/lib/timer/timer.c'

```
#include <kernel/timer.h>
#include <kernel/int.h>
#include <time.h>
void
timer (void)
{
    //
    // Conta le interruzioni.
    //
    os.timer.clocks++;
    //
    // Dovrebbe lanciare lo «schedulatore», ma qui non c'è;
    // pertanto, lancia direttamente delle applicazioni molto
    // brevi (devono garantire di terminare rapidamente).
    //
    ;
    ;
    ;
}
```

L'incremento della variabile ***os.timer.clocks*** consentirebbe di compiere delle azioni quando risulta trascorso un certo intervallo di tempo. Un'ipotesi di utilizzo potrebbe essere quella seguente, dove, ammesso che la frequenza del temporizzatore sia pari a '**CLOCKS_PER_SEC**', al trascorrere di ogni secondo fa qualcosa:

```
void
timer (void)
{
    os.timer.clocks++;
    if ((os.timer.clocks % CLOCKS_PER_SEC) == 0)
      {
        fa_qualcosa
      }
}
```

Gestione della tastiera

La gestione della tastiera è raccolta dalla libreria che fa capo al file di intestazione 'keyboard.h' come appare nel listato successivo.

Listato u9.108. '`./05/include/kernel/keyboard.h`'

```
#ifndef _KEYBOARD_H
#define _KEYBOARD_H     1

#include <kernel/os.h>

void keyboard (void);
void keyboard_load (void);

#endif
```

La funzione ***keyboard_load()*** definisce una mappa della tastiera, memorizzata negli array ***os.kbd.map1[]*** e ***os.kbd.map2[]***. Le due mappe riguardano i due livelli di scrittura: quello normale e quello che solitamente produce principalmente le maiuscole. L'indice degli array corrisponde al codice grezzo generato dalla tastiera (*scancode*). Il listato successivo riguarda una funzione ***keyboard_load()*** adatta alla disposizione italiana dei simboli, tenendo conto però che non si possono generare lettere accentate.

Listato u9.109. '`./05/lib/keyboard/keyboard_load.c`'

```
#include <kernel/keyboard.h>
void
keyboard_load (void)
{
    int i;
    for (i = 0; i <= 127; i++)
      {
        os.kbd.map1[i] = 0;
        os.kbd.map2[i] = 0;
      }
    //
    //
    //
    os.kbd.map1[1]   = 27;       os.kbd.map2[1]   = 27;
    os.kbd.map1[2]   = '1';      os.kbd.map2[2]   = '!';
    os.kbd.map1[3]   = '2';      os.kbd.map2[3]   = '"';
    os.kbd.map1[4]   = '3';      os.kbd.map2[4]   = 'L';        // 3, £
    os.kbd.map1[5]   = '4';      os.kbd.map2[5]   = '$';
    os.kbd.map1[6]   = '5';      os.kbd.map2[6]   = '%';
    os.kbd.map1[7]   = '6';      os.kbd.map2[7]   = '&';
    os.kbd.map1[8]   = '7';      os.kbd.map2[8]   = '/';
    os.kbd.map1[9]   = '8';      os.kbd.map2[9]   = '(';
    os.kbd.map1[10]  = '9';      os.kbd.map2[10]  = ')';
    os.kbd.map1[11]  = '0';      os.kbd.map2[11]  = '=';
    os.kbd.map1[12]  = '\'';     os.kbd.map2[12]  = '?';
    os.kbd.map1[13]  = 'i';      os.kbd.map2[13]  = '^';        // ì, ^
    os.kbd.map1[14]  = '\b';     os.kbd.map2[14]  = '\b';       // Backspace
    os.kbd.map1[15]  = '\t';     os.kbd.map2[15]  = '\t';
    os.kbd.map1[16]  = 'q';      os.kbd.map2[16]  = 'Q';
```

```
    os.kbd.map1[17]  = 'w';     os.kbd.map2[17]  = 'W';
    os.kbd.map1[18]  = 'e';     os.kbd.map2[18]  = 'E';
    os.kbd.map1[19]  = 'r';     os.kbd.map2[19]  = 'R';
    os.kbd.map1[20]  = 't';     os.kbd.map2[20]  = 'T';
    os.kbd.map1[21]  = 'y';     os.kbd.map2[21]  = 'Y';
    os.kbd.map1[22]  = 'u';     os.kbd.map2[22]  = 'U';
    os.kbd.map1[23]  = 'i';     os.kbd.map2[23]  = 'I';
    os.kbd.map1[24]  = 'o';     os.kbd.map2[24]  = 'O';
    os.kbd.map1[25]  = 'p';     os.kbd.map2[25]  = 'P';
    os.kbd.map1[26]  = '[';     os.kbd.map2[26]  = '{';         // è, é
    os.kbd.map1[27]  = ']';     os.kbd.map2[27]  = '}';         // +, *
    os.kbd.map1[28]  = '\n';    os.kbd.map2[28]  = '\n';        // Invio
    os.kbd.map1[30]  = 'a';     os.kbd.map2[30]  = 'A';
    os.kbd.map1[31]  = 's';     os.kbd.map2[31]  = 'S';
    os.kbd.map1[32]  = 'd';     os.kbd.map2[32]  = 'D';
    os.kbd.map1[33]  = 'f';     os.kbd.map2[33]  = 'F';
    os.kbd.map1[34]  = 'g';     os.kbd.map2[34]  = 'G';
    os.kbd.map1[35]  = 'h';     os.kbd.map2[35]  = 'H';
    os.kbd.map1[36]  = 'j';     os.kbd.map2[36]  = 'J';
    os.kbd.map1[37]  = 'k';     os.kbd.map2[37]  = 'K';
    os.kbd.map1[38]  = 'l';     os.kbd.map2[38]  = 'L';
    os.kbd.map1[39]  = '@';     os.kbd.map2[39]  = '@';         // ò, ç
    os.kbd.map1[40]  = '#';     os.kbd.map2[40]  = '#';         // à, °
    os.kbd.map1[41]  = '\\';    os.kbd.map2[41]  = '|';
    os.kbd.map1[43]  = 'u';     os.kbd.map2[43]  = 'U';         // ù, §
    os.kbd.map1[44]  = 'z';     os.kbd.map2[44]  = 'Z';
    os.kbd.map1[45]  = 'x';     os.kbd.map2[45]  = 'X';
    os.kbd.map1[46]  = 'c';     os.kbd.map2[46]  = 'C';
    os.kbd.map1[47]  = 'v';     os.kbd.map2[47]  = 'V';
    os.kbd.map1[48]  = 'b';     os.kbd.map2[48]  = 'B';
    os.kbd.map1[49]  = 'n';     os.kbd.map2[49]  = 'N';
    os.kbd.map1[50]  = 'm';     os.kbd.map2[50]  = 'M';
    os.kbd.map1[51]  = ',';     os.kbd.map2[51]  = ';';
    os.kbd.map1[52]  = '.';     os.kbd.map2[52]  = ':';
    os.kbd.map1[53]  = '-';     os.kbd.map2[53]  = '_';
    os.kbd.map1[56]  = '<';     os.kbd.map2[56]  = '>';
    os.kbd.map1[57]  = ' ';     os.kbd.map2[57]  = ' ';
    //
    os.kbd.map1[55]  = '*';     os.kbd.map2[55]  = '*';
    os.kbd.map1[71]  = '7';     os.kbd.map2[71]  = '7';
    os.kbd.map1[72]  = '8';     os.kbd.map2[72]  = '8';
    os.kbd.map1[73]  = '9';     os.kbd.map2[73]  = '9';
    os.kbd.map1[74]  = '-';     os.kbd.map2[74]  = '-';
    os.kbd.map1[75]  = '4';     os.kbd.map2[75]  = '4';
    os.kbd.map1[76]  = '5';     os.kbd.map2[76]  = '5';
    os.kbd.map1[77]  = '6';     os.kbd.map2[77]  = '6';
    os.kbd.map1[78]  = '+';     os.kbd.map2[78]  = '+';
    os.kbd.map1[79]  = '1';     os.kbd.map2[79]  = '1';
    os.kbd.map1[80]  = '2';     os.kbd.map2[80]  = '2';
```

```
    os.kbd.map1[81]  = '3';     os.kbd.map2[81]  = '3';
    os.kbd.map1[82]  = '0';     os.kbd.map2[82]  = '0';
    os.kbd.map1[83]  = '.';     os.kbd.map2[83]  = '.';
    os.kbd.map1[92]  = '/';     os.kbd.map2[92]  = '/';
    os.kbd.map1[96]  = '\n';    os.kbd.map2[96]  = '\n';        // Invio
}
```

La funzione ***keyboard()***, avviata ogni volta che si preme un tasto o lo si rilascia (attraverso l'impulso dato da IRQ 2), interpreta il codice grezzo proveniente dalla tastiera e aggiorna la variabile strutturata ***os.kbd***. Per esempio tiene traccia della pressione dei tasti [*Ctrl*], [*Alt*] e della selezione delle maiuscole. Quando si tratta di un tasto che deve produrre un carattere, questo viene annotato nella variabile ***os.kbd.key***, ma solo se questa è vuota. In pratica ci deve essere un programma che «consuma» questa informazione, azzerando di conseguenza la variabile. Si osservi che la variabile ***os.kbd.echo***, se contiene un valore diverso da zero, indica la richiesta di visualizzare sullo schermo ciò che si preme ed è controllata dalla macro-istruzione ***echo()*** (da '`vga.h`').

Listato u9.110. '`./05/lib/keyboard/keyboard.c`'

```
#include <kernel/keyboard.h>
#include <kernel/io.h>
#include <stdio.h>
void
keyboard (void)
{
    unsigned char scancode = inb (0x60);

    //
    // Shift, Shift-Lock, Ctrl, Alt
    //
    switch (scancode)
      {
        case 0x2A: os.kbd.shift = 1; break;
        case 0x36: os.kbd.shift = 1; break;
        case 0xAA: os.kbd.shift = 0; break;
        case 0xB6: os.kbd.shift = 0; break;
        case 0x1D: os.kbd.ctrl  = 1; break;
        case 0x9D: os.kbd.ctrl  = 0; break;
        case 0x38: os.kbd.alt   = 1; break;
        case 0xB8: os.kbd.alt   = 0; break;
        case 0x3A: os.kbd.shift_lock = ! os.kbd.shift_lock; break;
      }
    //
    // Ctrl+
    //
```

```
    if (scancode <= 127 && os.kbd.ctrl && os.kbd.key == 0)
      {
        switch (os.kbd.map1[scancode])
          {
            case 'a':  os.kbd.key = 0x01; break;  // SOH
            case 'b':  os.kbd.key = 0x02; break;  // STX
            case 'c':  os.kbd.key = 0x03; break;  // ETX
            case 'd':  os.kbd.key = 0x04; break;  // EOT
            case 'e':  os.kbd.key = 0x05; break;  // ENQ
            case 'f':  os.kbd.key = 0x06; break;  // ACK
            case 'g':  os.kbd.key = 0x07; break;  // BEL
            case 'h':  os.kbd.key = 0x08; break;  // BS
            case 'i':  os.kbd.key = 0x09; break;  // HT
            case 'j':  os.kbd.key = 0x0A; break;  // LF
            case 'k':  os.kbd.key = 0x0B; break;  // VT
            case 'l':  os.kbd.key = 0x0C; break;  // FF
            case 'm':  os.kbd.key = 0x0D; break;  // CR
            case 'n':  os.kbd.key = 0x0E; break;  // SO
            case 'o':  os.kbd.key = 0x0F; break;  // SI
            case 'p':  os.kbd.key = 0x10; break;  // DLE
            case 'q':  os.kbd.key = 0x11; break;  // DC1
            case 'r':  os.kbd.key = 0x12; break;  // DC2
            case 's':  os.kbd.key = 0x13; break;  // DC3
            case 't':  os.kbd.key = 0x14; break;  // DC4
            case 'u':  os.kbd.key = 0x15; break;  // NAK
            case 'v':  os.kbd.key = 0x16; break;  // SYN
            case 'w':  os.kbd.key = 0x17; break;  // ETB
            case 'x':  os.kbd.key = 0x18; break;  // CAN
            case 'y':  os.kbd.key = 0x19; break;  // EM
            case 'z':  os.kbd.key = 0x1A; break;  // SUB
            case '[':  os.kbd.key = 0x1B; break;  // ESC
            case '\\': os.kbd.key = 0x1C; break;  // FS
            case ']':  os.kbd.key = 0x1D; break;  // GS
            case '^':  os.kbd.key = 0x1E; break;  // RS
            case '_':  os.kbd.key = 0x1F; break;  // US
          }
        //
        if (os.kbd.echo && os.kbd.key)
          {
            (void) putchar (os.kbd.key);
          }
      }
    else if (scancode <= 127 && os.kbd.key == 0 && os.kbd.map1[scancode] != 0)
      {
        if (os.kbd.shift || os.kbd.shift_lock)
          {
            os.kbd.key = os.kbd.map2[scancode];
```

```
          }
        else
          {
            os.kbd.key = os.kbd.map1[scancode];
          }
        //
        if (os.kbd.echo && os.kbd.key)
          {
            (void) putchar (os.kbd.key);
          }
      }
}
```

Verifica del funzionamento

«

Per verificare il funzionamento delle chiamate di sistema, si può modificare il file '`kernel_main.c`' nel modo seguente. Ciò che si ottiene è di poter visualizzare sullo schermo il primo tasto che si preme (ciò avviene subito dopo la dichiarazione che il sistema è arrestato), in quanto non si possono inserire altri caratteri fino a quando «qualcuno» non svuota ***os.kbd.key***.

Figura u9.111. Modifiche da apportare al file '`./05/kernel/kernel_main.c`'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
#include <sys/syscall.h>
#include <kernel/timer.h>
#include <kernel/keyboard.h>
...
        //
        // Predispone la memoria libera per l'utilizzo.
        //
        mm_init ();
        //
        // Predispone il timer.
        //
        timer_freq (CLOCKS_PER_SEC);
        //
        // Predispone la tastiera.
        //
        keyboard_load ();
        echo ();
```

```
        //
        // Predispone la tabella IDT.
        //
        idt();
…
```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove dopo l'arresto dichiarato del sistema si può premere un tasto che viene visualizzato:

```
05 20070821183438
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010E45C  avail. 0010E45C..01EF0000
[kernel_memory_show] text   00100000..001054D8  rodata 001054E0..00105AC8
[kernel_memory_show] data   00105AC8..00105AC8  bss    00105AE0..0010E45C
[kernel_memory_show] limit  00001EF0
[gdt_print] base: 0x0010DB28 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000100000000010000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001000000000
[mm_init] available memory: 31333280 byte
[irq_remap] PIC (programmable interrupt controller) remap: ICW1, ICW2, ICW3,
ICW4, OCW1.
[kernel_main] system halted
h
```

Una specie di «shell»

«

Si conclude il lavoro del sistema giocattolo con una shell elementare, la quale deve acquisire i caratteri prodotti dalla tastiera e svolgere un compito in base al comando impartito. Ma prima di vedere il codice della funzione che svolge questo compito è necessario introdurre un'altra funzione, prevista dallo standard, che in precedenza è stata saltata: ***gets()***, dichiarata nel file di intestazione '`stdio.h`'.

La funzione ***gets()*** ottiene una stringa leggendo continuamente il contenuto della variabile '**`os.kbd.key`**'.

Listato u9.113. '`./05/lib/gets.c`'

```
#include <stdio.h>
#include <kernel/os.h>
char
*gets (char *s)
{
    int i;
```

```
    //
    // Legge os.kbd.char.
    //
    for (i = 0; i < 256; i++)
      {
        while (os.kbd.key == 0)
          {
            //
            // Attende un carattere.
            //
            ;
          }
        s[i] = os.kbd.key;
        os.kbd.key = 0;
        if (s[i] == '\n')
          {
            s[i] = 0;
            break;
          }
      }
    return s;
}
```

Realizzazione della shell

«

La shell è costituita dalla funzione ***shell()***, dichiarata nel file di intestazione '`app.h`', nel quale potrebbero essere inseriti i prototipi di altri tipi di applicazione, da avviare con l'aiuto della shell stessa.

Listato u9.114. '`./05/include/app/app.h`'

```
#ifndef _APP_H
#define _APP_H       1

void shell ();

#endif
```

Listato u9.115. './05/app/shell.c'

```
#include <app/app.h>
#include <stdio.h>
#include <string.h>
#include <kernel/gdt.h>
#include <kernel/kernel.h>
#include <kernel/mm.h>
#include <kernel/multiboot.h>
void
shell (void)
{
    char command[256];
    //
    //
    //
    while (true)
      {
        printf ("# ");
        //
        // Legge un comando.
        //
        gets (command);
        //
        if (strcmp (command, "quit") == 0
            || strcmp (command, "q") == 0)
          {
            break;
          }
        else if (strcmp (command, "help") == 0
                 || strcmp (command, "h") == 0)
          {
            printf ("shell commands:\n");
            printf ("h|help         = this help\n");
            printf ("q|quit         = quit the shell\n");
            printf ("i mb|info mb   = "
                    "show multiboot info\n");
            printf ("i gdt|info gdt = show gdt\n");
            printf ("i mem|info mem = show memory map\n");
```

```
            }
          else if (strcmp (command, "info mb") == 0
                   || strcmp (command, "i mb") == 0)
            {
              mboot_show ();
            }
          else if (strcmp (command, "info gdt") == 0
                   || strcmp (command, "i gdt") == 0)
            {
              gdt_print (&os.gdtr);
            }
          else if (strcmp (command, "info mem") == 0
                   || strcmp (command, "i mem") == 0)
            {
              kernel_memory_show ();
              mm_list ();
            }
          else
            {
              printf ("[%s] unknown command: %s\n", __func__,
                      command);
            }
        }
}
```

La shell mostra un invito e si spetta l'inserimento di comandi molto semplici, come '**`i mem`**' per avere la mappa dell'utilizzo della memoria. Se si sbaglia non è possibile correggere e la pressione di tasti per la cancellazione provoca semplicemente la scrittura di codici non gestiti. Si osservi che anche gli spazi superflui contano come «errori».

Conclusione

Per concludere viene mostrato il listato definitivo del file '`kernel_main.c`', in cui si avvia la shell. Se con questo sistema si volesse fare qualcosa di più, basterebbe intervenire nella shell stessa, senza ritoccare ulteriormente il file '`kernel_main.c`'.

Listato u9.116. './05/kernel/kernel_main.c'

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>
#include <sys/syscall.h>
#include <kernel/timer.h>
#include <kernel/keyboard.h>
#include <app/app.h>
//
// Funzione principale, da dove si avvia il kernel.
//
void
kernel_main (unsigned long magic, multiboot_t *info)
{
    //
    // Inizializza i dati relativi alla gestione dello
    // schermo VGA, quindi ripulisce lo schermo.
    //
    vga_init ();
    clear ();
    //
    // Data e orario di compilazione.
    //
    printf ("05 %s\n", BUILD_DATE);
    //
    // Cerca le informazioni «multiboot».
    //
    if (magic == 0x2BADB002)
      {
        //
        // Salva e mostra le informazioni multiboot.
        //
        mboot_info (info);
        mboot_show ();
```

```
      //
      // Raccoglie i dati sulla memoria fisica.
      //
      kernel_memory (info);
      //
      // Predispone la tabella GDT.
      //
      gdt ();
      //
      // Predispone la memoria libera per l'utilizzo.
      //
      mm_init ();
      //
      // Predispone il timer.
      //
      timer_freq (CLOCKS_PER_SEC);
      //
      // Predispone la tastiera.
      //
      keyboard_load ();
      echo ();
      //
      // Predispone la tabella IDT.
      //
      idt();
    }
  else
    {
      printf ("[%s] no \"multiboot\" header!\n",
              __func__);
    }
  //
  // Shell.
  //
  shell ();
  //
  printf ("[%s] system halted\n", __func__);
  _Exit (0);
```

```
}
```

Nella schermata successiva si vede una breve interazione con la shell, dove appare anche un errore di digitazione.

```
# help
shell commands:
h|help          = this help
q|quit          = quit the shell
i mb|info mb    = show multiboot info
i gdt|info gdt = show gdt
i mem|info mem = show memory map
# info mb
[mboot_show] flags: 00000000000000000000011111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
# info gdt
[gdt_print] base: 0x0010E068 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 00000000010000000001000000000000
[gdt_print] 1 00000000000000000001111011110000 00000000110000001001101000000000
[gdt_print] 2 00000000000000000001111011110000 00000000110000001001001100000000
# info em...
[shell] unknown command: info em...
# info mem
[kernel_memory_show] kernel 00100000..0010E5A4  avail. 0010E5A4..01EF0000
[kernel_memory_show] text   00100000..001058F4  rodata 00105900..0010600C
[kernel_memory_show] data   0010600C..0010600C  bss    00106020..0010E5A4
[kernel_memory_show] limit  00001EF0
[mm_list] free 0010E5A8..01EF0000 size 01EEFFFC
# quit
[kernel_main] system halted
```

[1] Evidentemente, il limite massimo teorico della frequenza che può essere generata è proprio 1,193181 MHz che si ottiene dividendo semplicemente per uno; inoltre, sapendo che il divisore può avere al massimo il valore 65535, la frequenza minima è di 18,22 Hz, corrispondente al valore predefinito iniziale.